Friends and Foes in Monitoring APIs

OSMC, Nürnberg, 2016-11-30

Heinrich Hartmann, Circonus

Hi, I am Heinrich

heinrich.hartmann@circonus.com

@HeinrichHartman(n)

- From Mainz, now in Munich
- PhD in Mathematics (Bonn)
- Independent IT Consultant
- Analytics Lead at Circonus,

  Monitoring and Analytics Platform

CIRCONUS

Circonus in a Nutshell

- Monitoring and Analytics Platform
- SaaS and on-Premise Versions
- Founded in 2010 by Theo Schlossnagle (OmniTI)
- Build by Engineers for Engineers

  OmniOS (zfs/zones) / Inhose-DB / Inhouse-Q / ...
- Highly scalable and cost efficient
- Advanced Analytics (Histograms, Forecasting, AD, OD, CAQL)
- Trusted by: Groupon, Cisco/WebEx, Joyent, Hashicorp, ...

Open Source Contributions

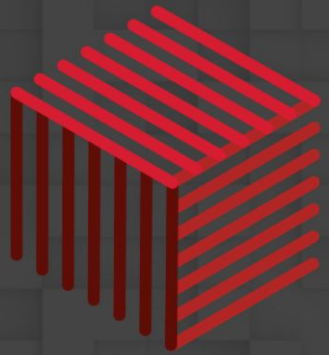
https://github.com/circonus-labs

- NAD https://github.com/circonus-labs/nad

  Lightweight, 0-conf, extensible agent that collects *correct* data, supports reverse-pull.
- Reconnoiter https://github.com/circonus-labs/reconnoiter

  Large scale metric collection service (“broker”)

- fq https://github.com/circonus-labs/fq - fast relieable pub-sub messaging
- libmtev https://github.com/circonus-labs/libmtev - Event-driven application framework
- libcirclhist https://github.com/circonus-labs/libcircllhist - Circonus’ Histograms
- Instrumentation packages, ...



IRONdb

POWERED BY CIRCONUS

BETA

IRONdb
POWERED BY CIRCONUS

IRONdb.io

For a Durable, Scalable Graphite Infrastructure

A scalable time-series backend for graphite

Add IRONdb:

NO more shutting off and deleting metrics
NO more setting up cold standbys
NO learning curve

>> Add IRONdb for a Fault Tolerant experience <<

irondb.io

BETA

IRONdb
POWERED BY CIRCONUS

IRONdb.io

For a Durable, Scalable Graphite Infrastructure

25,000 metrics
FREE

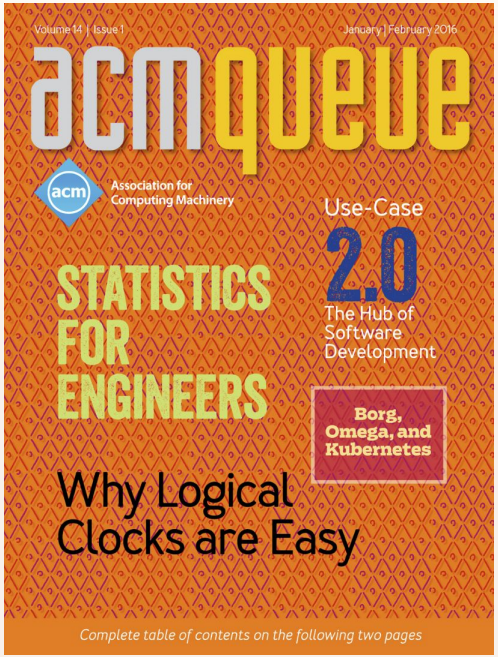
100,000 metrics
$10k

100,000,000 metrics
$1MM

irondb.io

BETA

I have been talking about #StatsForEngineers



[1] Statistics for Engineers @ ACM Queue and @ CACM

[2] Statistics for Engineers Workshop Material @ GitHub

[3] Spike Erosion @ circonus.com

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

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

Talks

[5] Workshop “Statistics for Engineers” @ SREcon16 | USENIX in Dublin

[6] Monitorama PXD 2016

[7] Netways/OSMC Nürnberg 2016 (upcoming)

A tale of API Monitoring

Use Case: “Xalando” - a fashion webstore

- A (fictional) webstore for fashion products
- 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

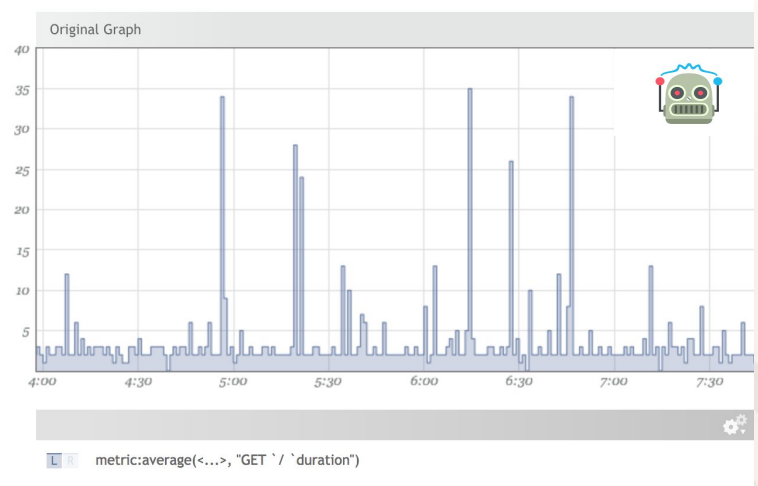
1. Make a synthetic request every minute
2. Measure and store request latency

Good for

- Measure Availability
- Alert on outages

Bad for

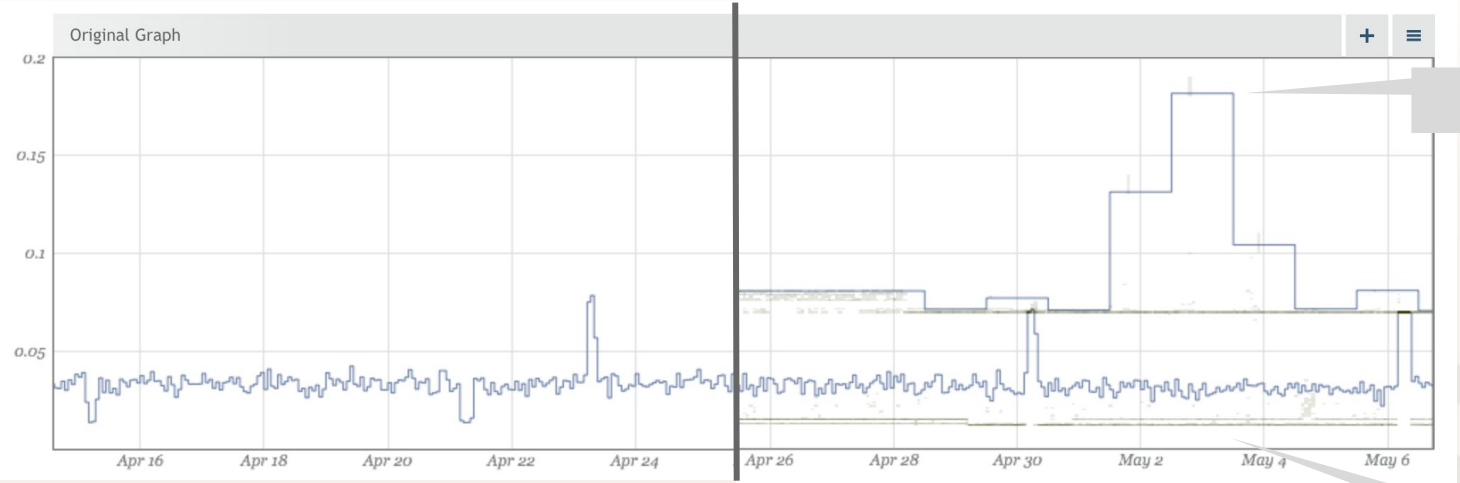
- Measuring user experience



Latencies of synthetic requests over time

CIRCONUS

<!> Spike Erosion </!>



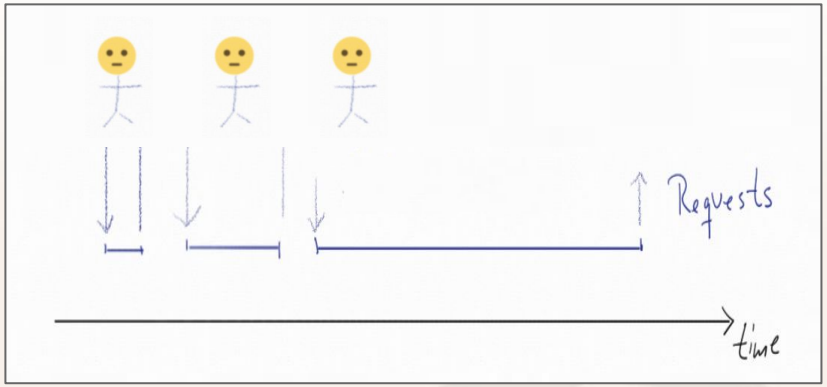
- On long time ranges, aggregated / rolled-up data is commonly displayed
- This practice "erodes" latency spikes heavily!
- Store all data and use alternative aggregation methods (min/max) to get full picture, cf. [3].

{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.

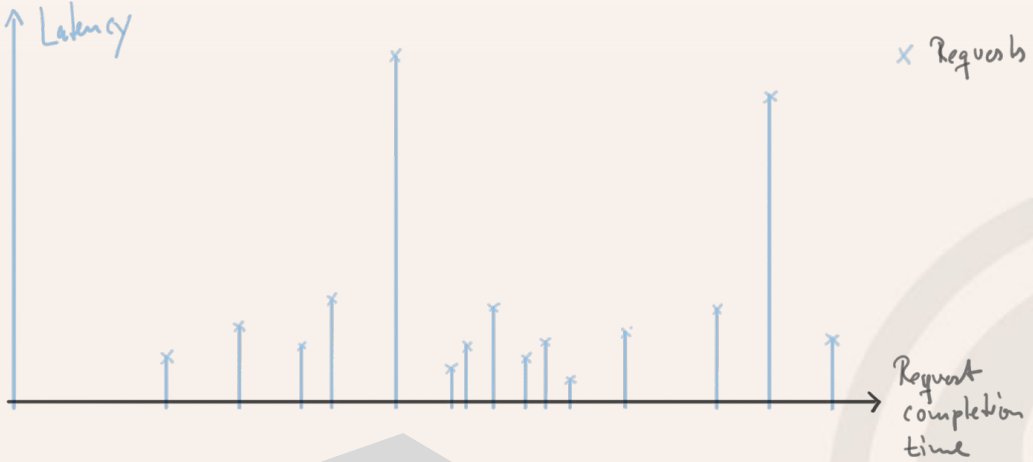
Discussion

- 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

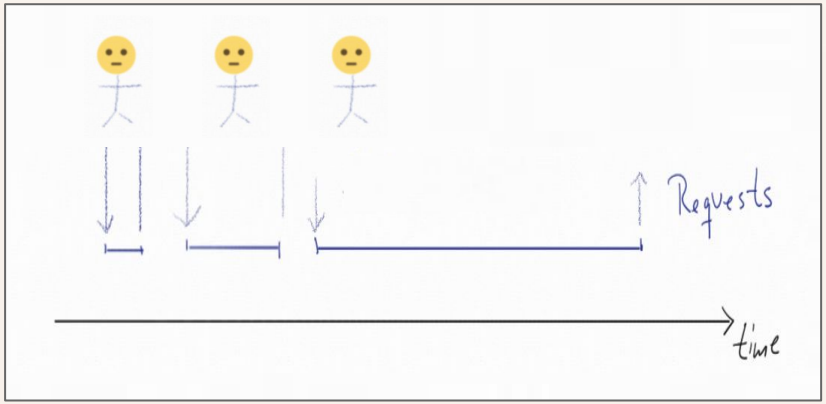
a concise visualization of the API usage

Latency on the y-axis

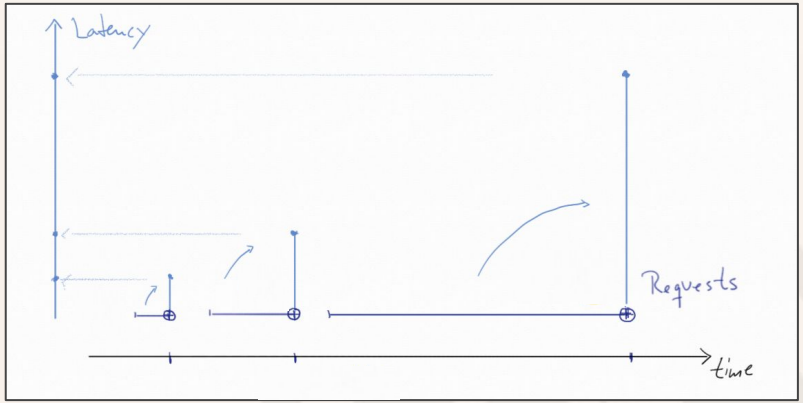


time the request was completed

Construction of the Request-Latency Chart (RLC)



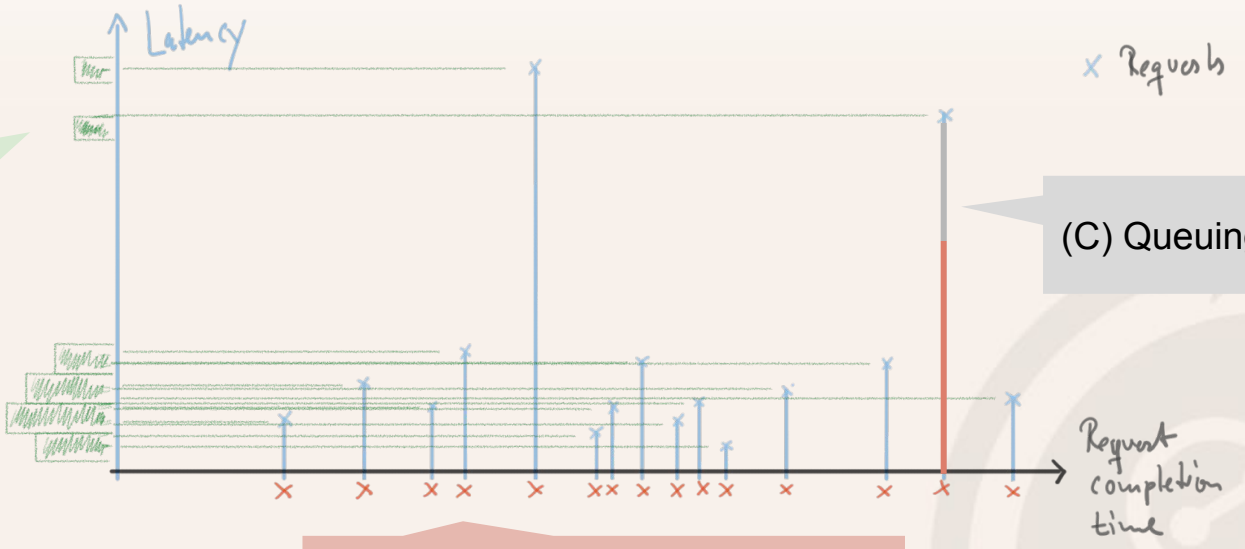
Request Latency UML Diagram



Request Latency Chart

Math view on APIs

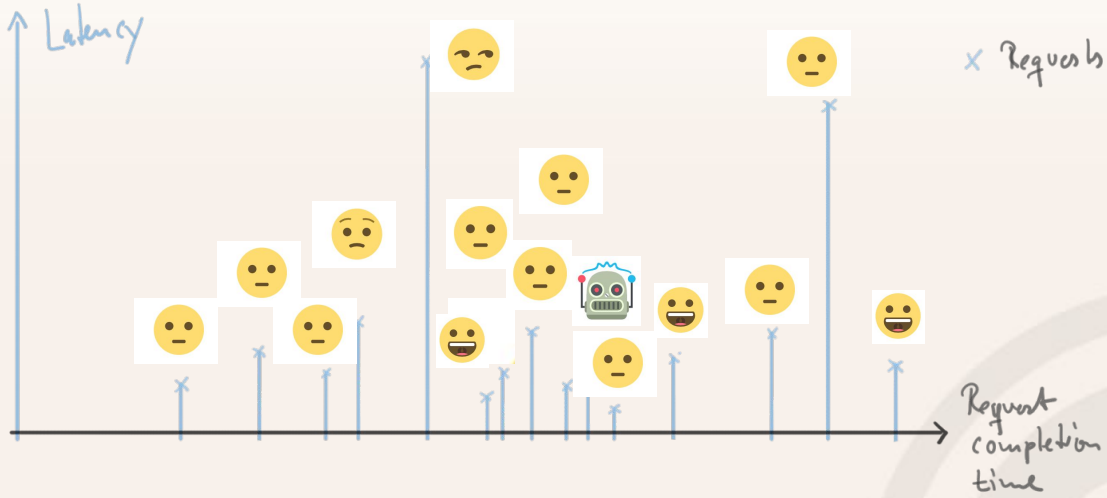
(A) Latency distribution



(C) Queuing theory

(B) Arrival/Completion times

“Requests are People”

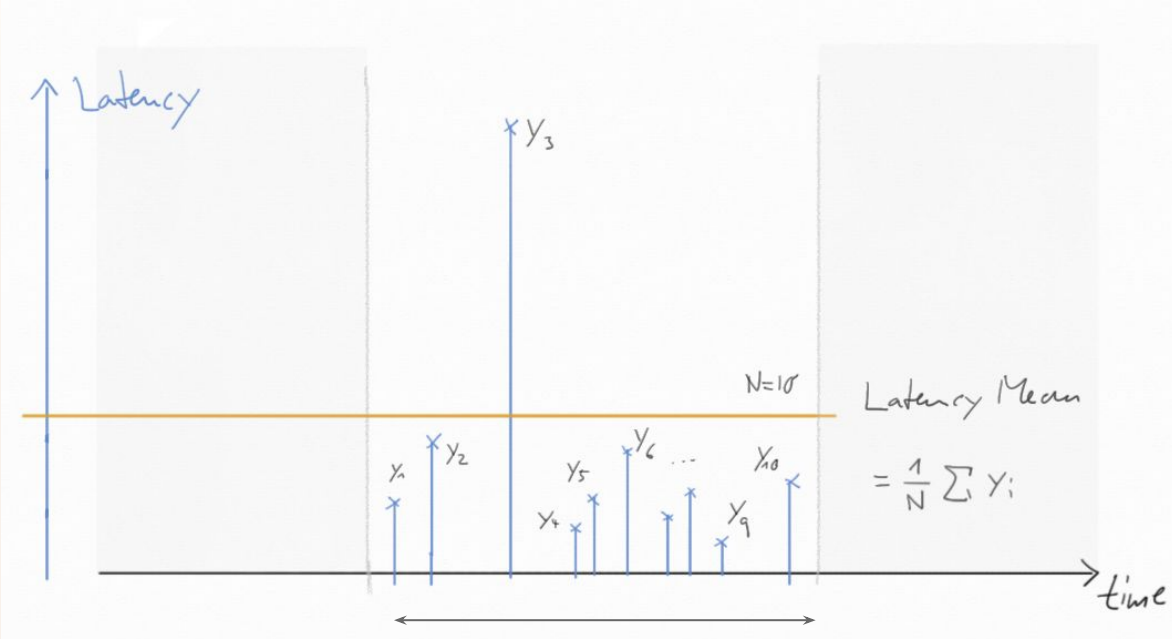


If you care about your users, you care about their requests.

Every single one.

{3} Monitoring Latency Averages

{3} What are latency mean values?



reporting period

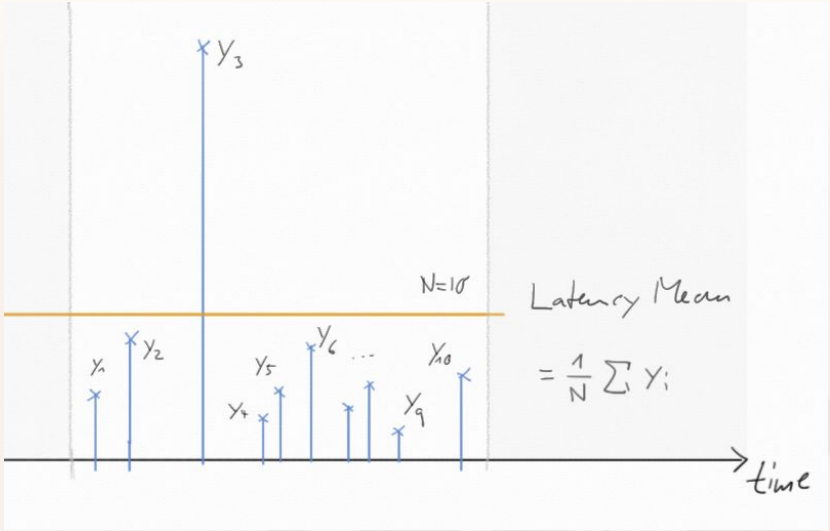
{3} Mean Request Latency Monitoring

Method

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

Pro/Con

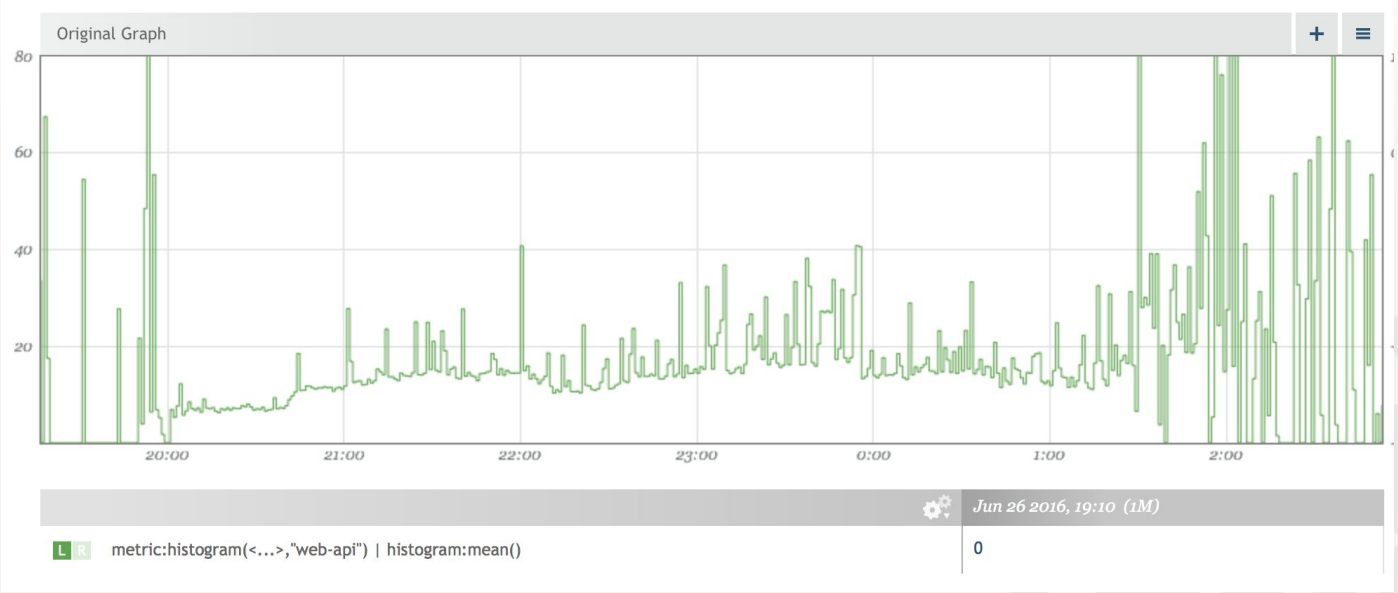
+ Measure requests by actual people
+ Cheap to collect store and analyze
- Easily skewed by outliers at the high end (complex, long running requests)
- ... and the low end (cached responses)



"Measuring the average latency is like measuring the average temperature in a hospital."

-- Dogan @ Optimizely

{3} Mean Request Latency in practice

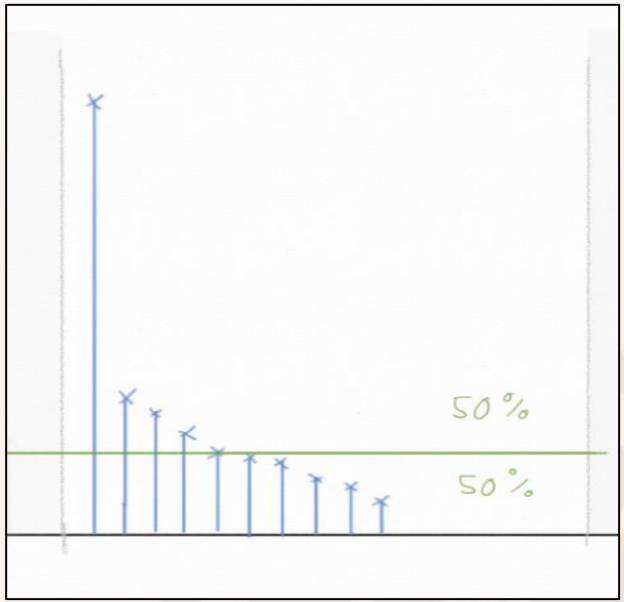


{3} Mean Request Latency - Robust Variants

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

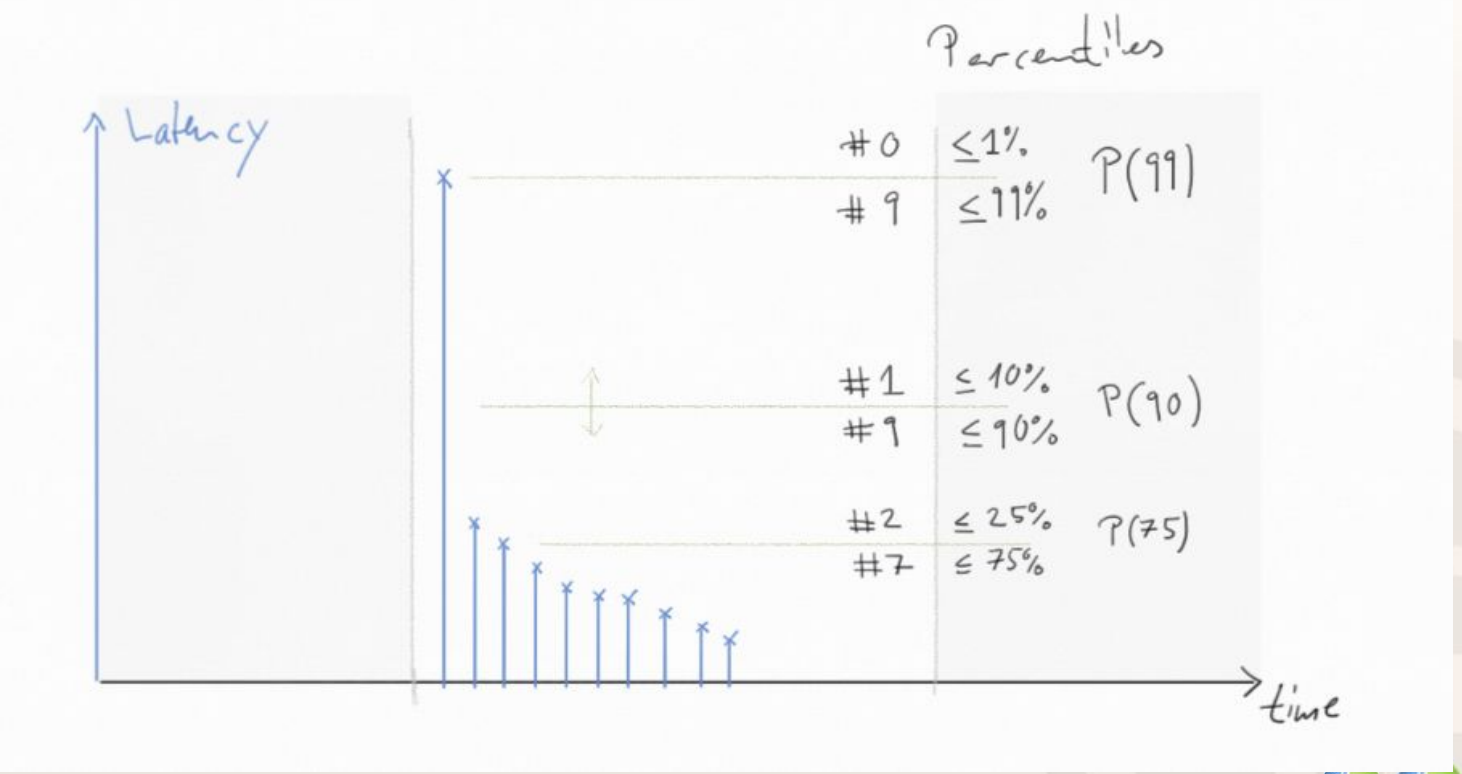
3. Collect Deviation Measures
   - Avoid standdard deviations, use
   - Use Mean absolute deviation



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

CIRCONUS

<!> 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

CIRCONUS

An Argument with John Rauser@Snapchat

CIRCONUS

Dear John,

Thank you very much for your comments. I appreciate your passion for this topic. Percentiles are a delicate and subtle topic. It's great to have this conversation. I was not able to put my remarks into tweets, so I am using this old fashioned "letter" form to reply.

So, I said:

"Percentiles can't be aggregated!"

This quote is a compressed version of the following statement:

> Mean values of percentiles are not percentiles of a larger population. Furthermore, there is no aggregation method that gives you total percentiles from percentiles of subsets of your data.

This addresses a mis-perception that is paramount in the monitoring community. Take for example this graph I just found on Google (source):

I am pretty sure, that this is not the the 95%-tile of the ~2h represented by each point. It's instead the average of all 95%-tiles measured over 5M intervals. This is something different. And you should be aware of that. Most people are not. This is my point!

Can we fix it, by using another aggregation method? No way. Sorry.

Can some information be derived from averaged percentiles? Well, yes. But it's _super_ subtle.

Can we do better? Yes certainly! Store and aggregate Histograms. At Circonus we spent so much time and resources to make this work for our users. WE CARE ABOUT GETTING THIS RIGHT! It's super frustrating to see people being mislead by their tools and not even noticing it. And by the way, Google's John Banning, who presented Google's 'Global Monitoring System' directly after me referenced my presentation multiple times for this point. At Google they do care as well, and use histograms!

So, Is the initial quote bold and compressed. Well yes, of course! I am on a stage in front of 300 people that want to be entertained ;)

I just found a wonderful article about the art of presentation (https://thefractionatingcolumn.com) that makes some great points: Story, Content, Passion, Humor. I fully agree with those points, and I consciously made the decision to sacrifice some precision to "make it sticky".

Sincerely,
Heinrich Hartmann, Portland, 2016-06-30

You CAN average percentiles!

John Rauser

July 2016

… at least some of the time.

"You can't meaningfully average percentiles."

From time to time I hear people say "You can't meaningfully average percentiles." This has always irritated me. Last week I said as much on twitter.

This document is an attempt to explain my position. It is also an attempt to teach statistical thinking by example, and to demonstrate using the R language for data analysis. This document was written in RMarkdown, and you can download the source here and play with it yourself.

If the authors linked above had said "In an operational context, averaging percentiles is a bad idea," or more simply, "Averaging percentiles is very dangerous," I would have no quarrel, but by making such extreme statements they shut down opportunities for statistical thinking, and as someone who cares a great deal about statistical education, that makes me sad.

The average is just a statistical tool. Like any tool it can be used wisely or foolishly. You can average any data you like, and the average is always "meaningful" in that it has well understood mathematical properties. Whether or not an average is *useful* depends on how your data was generated and what claims you're making about that average.

What is the analytical task at hand?

What the average of percentiles is *not* is a percentile. It is certainly true that there is no way to recover the *exact* population percentile from a collection of sample percentiles. When the authors linked above say that the median of the sample medians is not the population median, they are completely correct.

But when trying to summarize a data set, the question you should always be asking yourself is **what is the analytical task at hand**?

The authors above come from the world of operating large fleets of computers (I world I spent many years in), so I'll choose examples from that realm. Let's say that you have a set of machines, each of which has computed its own 90th percentile of service latency over some span of time (an hour, for example), and you want to know the overall 90th percentile across all the machines. To get the exact 90th percentile, you would need to examine all the raw latency data from all of the machines. This might be deemed too difficult an engineering feat, or it may well be that the raw data is just gone, and all you have is the 90th percentiles from each host.

Now the truth is that you never really needed to know the exact 90th percentile. You were always willing to settle for an estimate with some small amount of error. (I hope this is true, because in a distributed computing environment that is all you can ever achieve.) Further, let's say you're willing to assume that each of the machines in your fleet is functionally the same, and that the observations are distributed across the fleet randomly, with uniform probability; in technical language, your data is independent and identically distributed (often abbreviated i.i.d.). In many real fleets, most of the time, this assumption is completely reasonable. Later on we'll examine what happens when this assumption is violated.

The question then is: **Is the average of the 90th percentiles a good estimator of the overall 90th percentile?** 'Goodness' in this context has a technical definition in statistics, but a big concern is bias. It's probably a STAT301 exercise to prove that in the general case the average of the percentiles is not an unbiased estimator and is therefore (some might conclude) "bad". But how bad is it? How does it break down?

We could do calculus and prove theorems, but that's no fun. We can code, let's simulate!

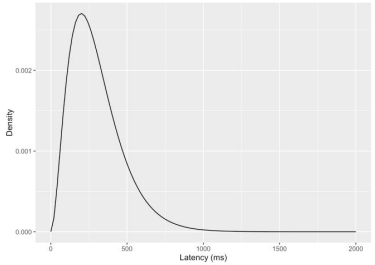
Simulation

We said we were looking at latency data. Latency data is often roughly gamma distributed, so let's pretend that a gamma process generates our data. Here's one such process, gamma(3, 1/100):

```
# Generate a dataset with 100 points from [0, 2000]
to_plot<-data.frame(x=seq(0,2000,length.out=100)) %>%
# ... and set y to the density of the gamma distribution at each point
  mutate(y=dgamma(x,3,1/100))

# Plot the curve
ggplot(to_plot, aes(x,y)) + geom_line() +
```

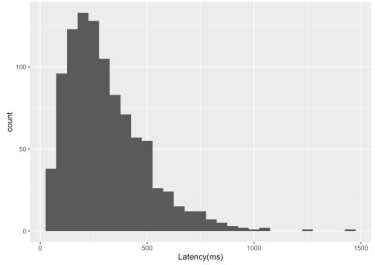
```
# Plot the curve
ggplot(to_plot, aes(x,y)) + geom_line() +
  xlab("Latency (ms)") + ylab("Density")
```



In a purely ideal world, if your service has to do three sub-tasks in sequence, and each of sub-tasks has exponentially distributed latency with an average latency of 100ms, this would be the latency distribution of your service.

Let's draw a sample of 1,000 data points from this ideal distribution and plot a histogram with 50ms bins.

```
# Draw the sample
to_plot<-data.frame(latency=rgamma(1000, 3, 1/100))
# ... and plot a histogram
ggplot(to_plot, aes(latency))+geom_histogram(binwidth=50)+xlab("Latency(ms)")
```



So our example service has a latency distribution that is skew, with a mean of 300 milliseconds, and a right tail that stretches out and is sometimes over 1,000 ms. The true 90th percentile of this process is around 532 ms.

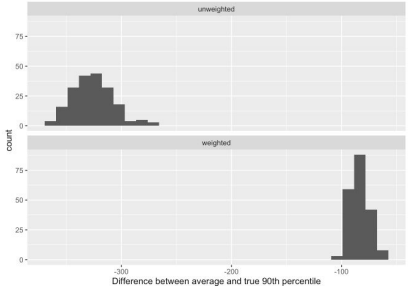
```
qgamma(0.9, 3, 1/100)
```

…

```
# Draw the replicates
reps<-map_df(1:200, do_one_replication,
             simf=simulate_one_day_varying, rate_multiplier=1)

# Reshape the data for faceted plotting
to_plot<-reps %>%
  select(-idx) %>%
  gather(kind, value, -true_p90)

# Plot faceted histograms
ggplot(to_plot, aes(value-true_p90)) + geom_histogram(bins=30) +
  facet_wrap(~kind, ncol=1) +
  xlab("Difference between average and true 90th percentile")
```



Now the unweighted average is very far off and even the weighted average is really starting to suffer.

Percentile ranks and histograms

A nice way to sidestep this whole issue is to reason in terms of percentile ranks and not percentiles. If you're willing to put a line (or lines) in the sand up front and say that certain latency thresholds are important – if you think (for example) that 1,000ms is a key threshold, then for any subset of requests you can just record the number of requests and the number over 1,000ms. Those subsets can be trivially combined to compute a combined percentile rank.

If you're not willing to draw those lines in the sand, and you can afford the storage, you can also keep track of (approximate) histograms which you can combine cheaply to compute (approximate) percentiles across fleets or over long spans of time.

Conclusion

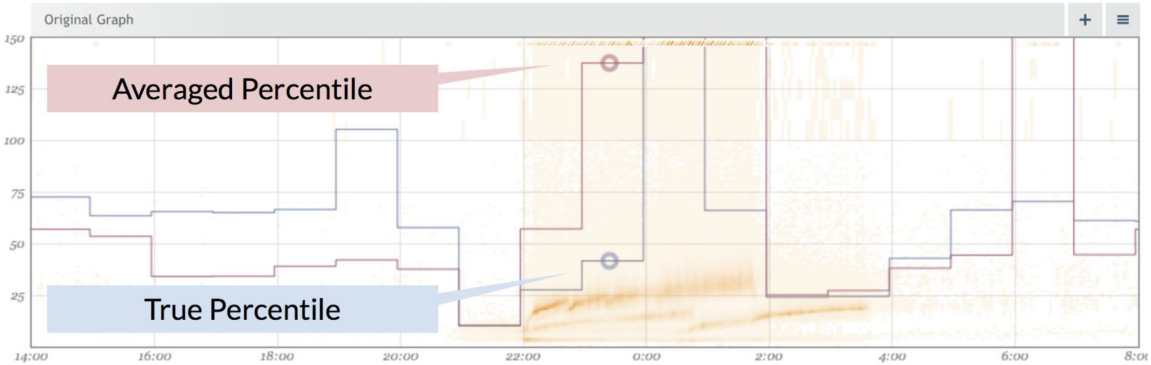
I hope that I've given you some intuition about how the process of averaging summary statistics like percentiles works, when it's appropriate, and when it breaks down.

But more, I hope I've shown you how easy it is to simulate statistical processes in order to gain an understanding of whether and how they work. Anytime a supposed expert (myself included) tells you what you can or can't do with your data, put on your skeptic's hat, code up your own simulations on your data, and see for yourself. This is major part of how I taught myself statistical intuition, and I suspect it will work for you too.

http://rpubs.com/jrauser/percentiles

CIRCONUS

Don’t aggregate percentiles!



In reply to John Rauser

Heinrich Hartmann @heinrichhartman · Jul 5

@jrauser Nice writeup. In practice things look pretty bad, though...

1 1

CIRCONUS

{5} Desireable Properties for Metrics

A statistic S **robust** if S(A) if a small number of (severe) outliers induces a small change in S(A).

A statistic S is **mergable** if S(A ∪ B) can be computed from S(A) and S(B).

{5} It looks like you can only have one:

Type	Statistic	Robust	Mergable
Centrality	mean		X
	median	X	
Deviation	stddev		X
	IQR	X	
Distribution	Percentiles	X	

{5} API Monitoring with Histograms

{5} API Monitoring with Histograms

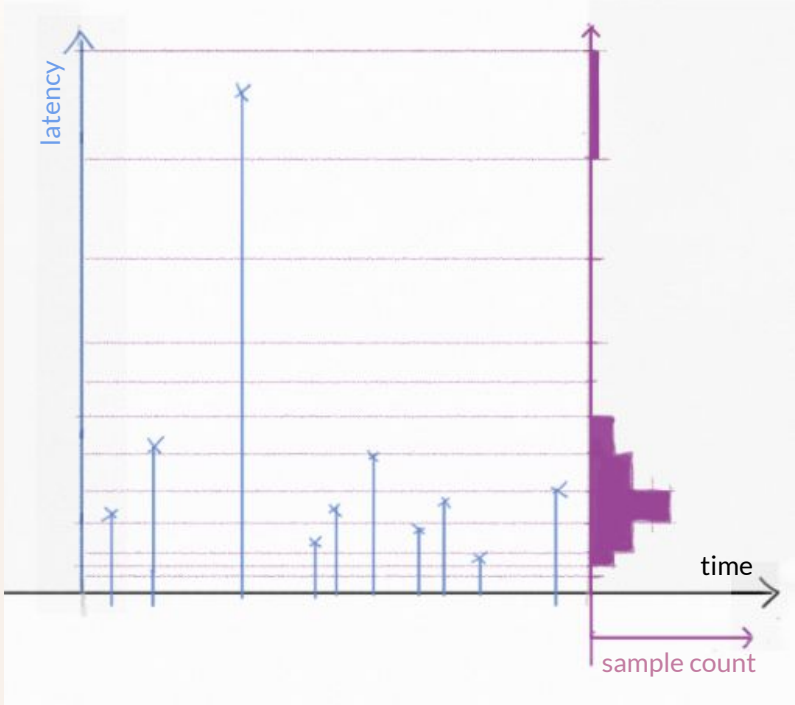
Method

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



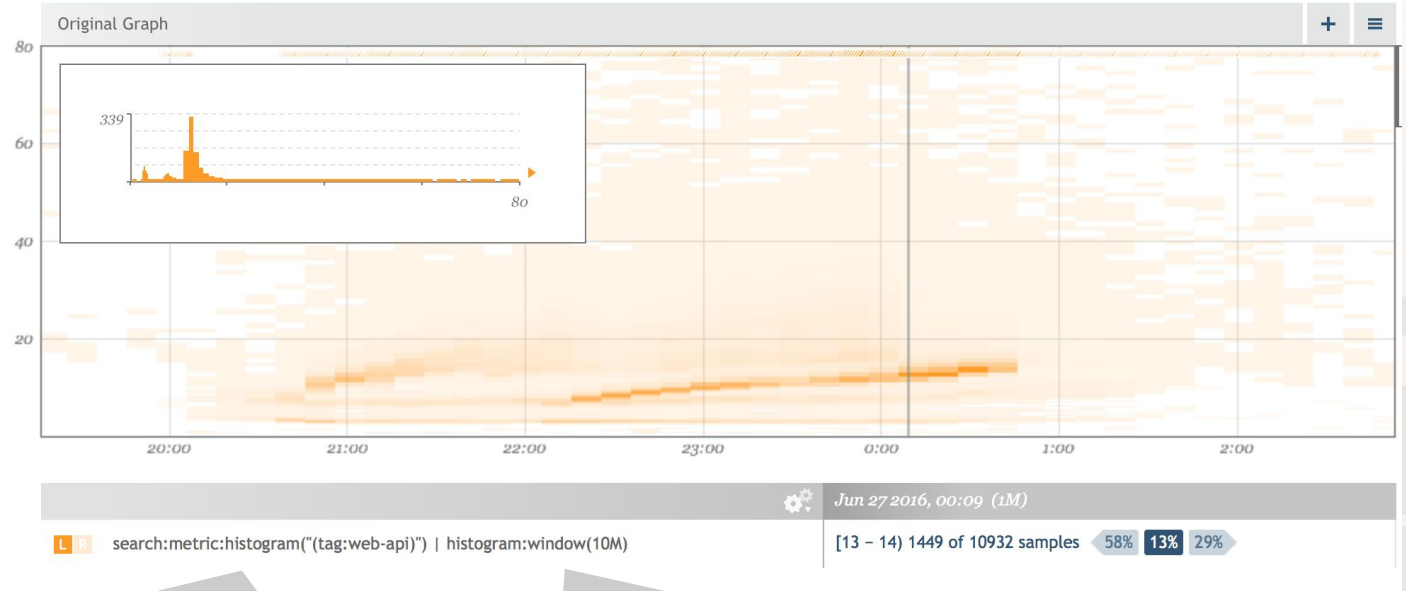
CIRCONUS

Histograms as Ultimate Summary Statistic

- Histograms are robust.
- Histograms are mergable.
- Histograms are an univesersal mergable enhancement

{5} Histogram Monitoring in Practice

Histograms can be visualized as heatmaps.



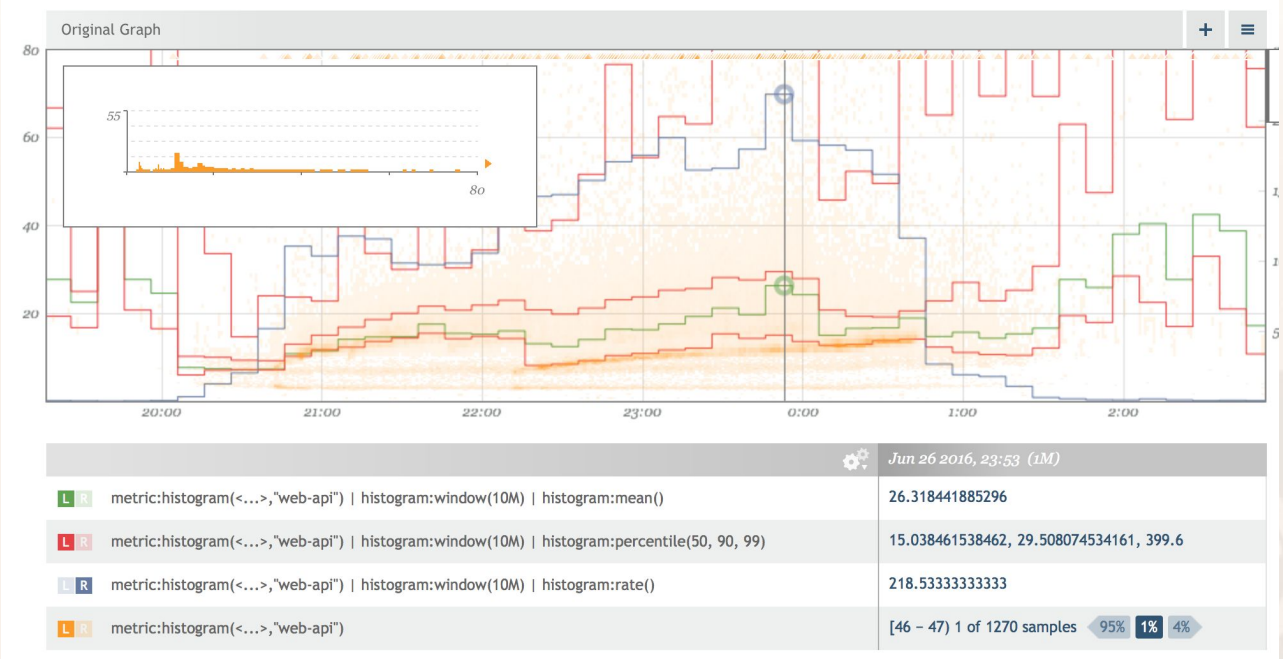
Aggregate data from all nodes serving “web-api”

.. across windows of 10min.

CIRCONUS

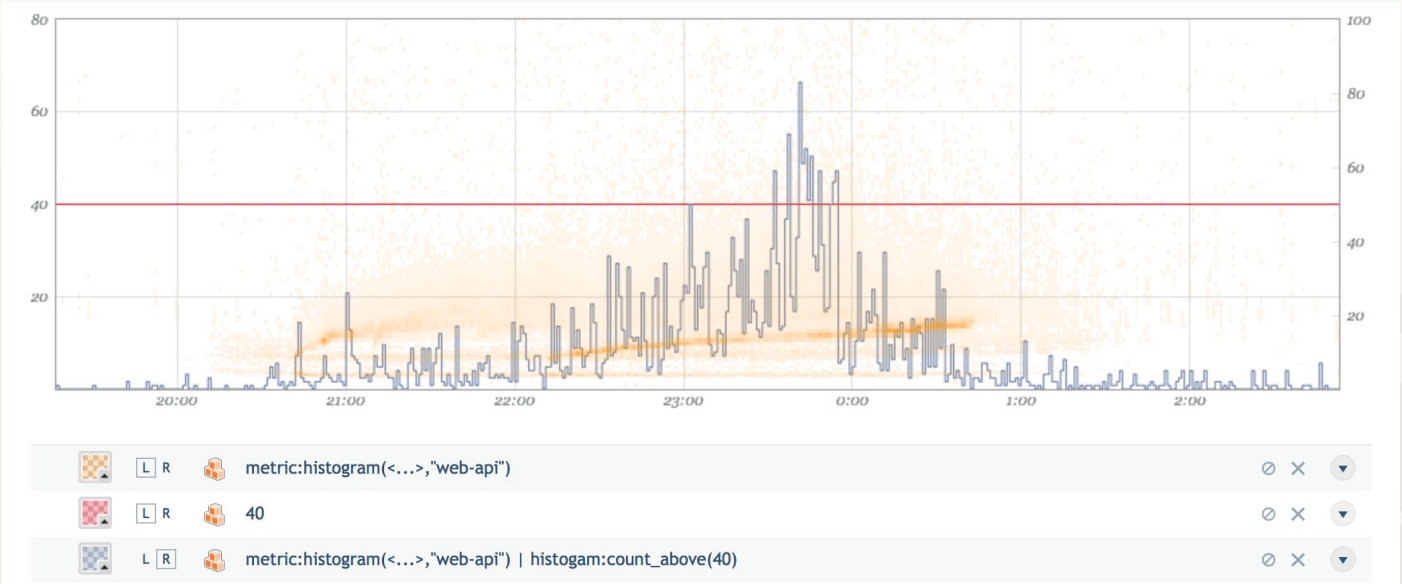
{5} Histogram Monitoring in Practice

All kinds of metrics can be derived from histograms



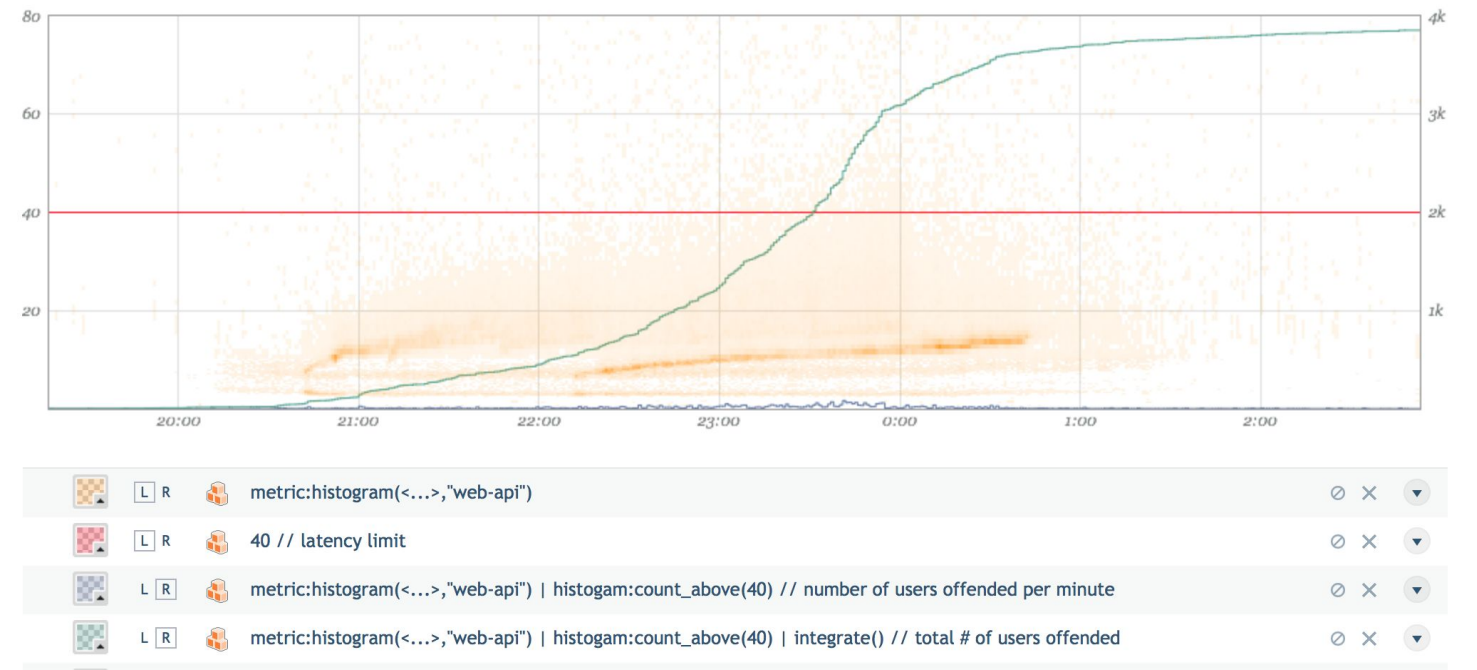
{6} The search for meaningful metrics

{6} Users offended per minute



CIRCONUS

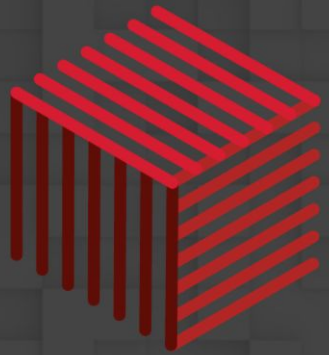
{6} Total users offended so far



CIRCONUS

Takeaways

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



IRONdb

POWERED BY CIRCONUS

BACKUP