For any API you care about ...

How many requests in August were served within 100ms?
For any API you care about ...

How many requests in August were served within 150ms?
For any API you care about ...

How many requests in August were served within $180\text{ms}$?
For any API you care about ...

How many requests in **June 16th 9:12-9:35** were served within 100ms?
Hi, I am Heinrich

- Data Scientist at Circonus
- PhD in Mathematics
- Talks about #StatisticsForEngineers
- Lives in Stemwede, Germany
Agenda

- Why monitor Latency?
- What is an SLO?
- Three methods to calculate Latency SLOs
  - Method 1
  - Method 2
  - Method 3
- Conclusion
Why monitor Latency?

Latency is a key performance indicator for any API.

Four Golden Signals from the SRE Book:

> “Latency, Traffic, Errors, and Saturation”

This later became the RED Method:

Requests, Errors, Duration.
What is an SLO?

Service Level Indicator (SLI)
A metric that quantifies the quality or reliability of your service. Typically of the form “Good Events / Valid Events * 100”.

Service Level Objective (SLO)
A target value for the service level, as measured by an SLI, that sets expectations about how the service will perform.

Service Level Agreement (SLA)
What happens if a published SLO is not met?
Availability

SLI

Once a minute, ssh into the target host, report 1 if it’s working 0 if not.

SLO

Uptime 99.9%, i.e. SLI = 1 over the last month 99.9% of the time.

SLA

If we don’t meet our SLO in one month, you will get exactly one cake.
Latency SLOs Example from SRECon 2018*

SLI

“The proportion of valid requests that were served within < 1s”

SLO

“99% of valid requests in the past 28 days served in < 1s”

SLA

(*) SRECon 2018 : Developing Effective ... Fong-Jones, Bennett, Quinlan, Stockman, Thorne @ Google Script, p16, we replaced “100ms” by “1s” for the example.
Was the SLO met?

SLO: 99% of valid requests in the past 28 days served in <1s

Source: Developing Effective ... Fong-Jones, Bennett, Quinlan, Stockman, Thorne @ Google, Script, p26
Was the SLO met?

SLO: 99% of valid requests in the past 28 days served in <1s

What if I told you, 99.5% of all requests occurred here?

Source: Developing Effective ... Fong-Jones, Bennett, Quinlan, Stockman, Thorne @ Google, Script, p26
Was the SLO met?

SLO: 99% of valid requests in the past 6h served in <50ms

Source: Circonus, internal
Percentile Metrics can’t be used for SLOs

For SLOs we need to compute percentiles over ...

(1) multiple weeks of data

(2) multiple nodes (potentially).

But: **Percentiles can’t be aggregated.**
Percentiles can’t be aggregated

Long story of discussions following my Monitorama 2016 talk:

Dan Slimmon (@danslimmon)

_counts cannot be aggregated_

— @heinrichhartman #monitorama

1:23 AM · 30 Jun 2016 from Portland, OR

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:

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.
Percentiles can’t be aggregated

Don’t aggregate percentiles!

Averaged Percentile

True Percentile

> 300% off

John Rauser
@jrauser

Replying to @heinrichhartman

Yep. It’s easy to find real world examples where what seems like a good idea fails.

3:32 AM - 5 Jul 2016
Averaging API Latency Percentiles over 24 hours

API Latency Distribution over 24h

p90 35.8ms averaged p90 60.3ms (68.5% error)

811K Requests
Latency SLOs Done Right

**Task**

Count all requests over $period served faster than $threshold.

**Three valid Methods**

1. Log data
2. Counter Metrics
3. Histogram Metrics
(1) Latency SLOs via Log Data

SELECT count(*) FROM logs
WHERE
  time in $period
AND
  latency < $threshold
(1) Latency SLOs via Log Data

+ Correct, Clean, Easy

- You need to keep all your log data for months
- ... which can get very expensive.

You can do this with: ssh+awk, ELK, Splunk, Honeycomb, etc.
(2) Latency SLOs with Counter Metrics

- Pick a latency $threshold, e.g. 1s

- Add a metric that counts how many requests were faster than $threshold

- ... store as e.g. “aws-eu.www22.GET./.lt_1s”

- ... for each node and endpoint you care about

- Sum / Integrate these metrics across nodes, endpoint and time:

  \[
  \text{find(“aws-eu.www*.GET.*.lt_1s”) | stats:sum() | integrate()}
  \]
(2) Latency SLOs with Counter Metrics

9.5 K / 106 K = 8.9%

Slow Requests
(2) Latency SLOs via Counter Metrics

+ Easy, Correct
+ Cost effective
+ Full flexibility in choosing aggregation intervals
- Need to choose (multiple) latency thresholds upfront

You can do this with:
  Prometheus ("Histograms"), Graphite, DataDog, VividCortext
HDR Histogram Data Structures

Store Latency distribution in sparse, log-linear histogram data structures:

- Total of 46,081 bins that cover the decimal float range $\pm 10^{\pm 128}$ (HDR)
- One bin for each decimal float number with two significant digits (log-linear):
  10, 11, 12, ... 100, 110, 120, ..., 1000, 1100, 1200,...
- Only store bins which have entries (sparse encoding)
- Typical size 300b / Histogram even with >100K entries

Open Source Implementations:
- hdrhistogram.org (Tene @ Azul Systems)
- github.com/circonus-labs/libcircllhist (Circonus, IronDB)
(3) Latency SLOs with Histograms

1. Capture latency information of all relevant APIs as histogram metrics.

2. Aggregate latency histograms over nodes, endpoints and time. Get total latency distribution over SLO timeframe (weeks, months).

3. Count samples in bins below the thresholds to compute SLOs.
89.3% of all requests in August 2018 were served within 50ms.
(3) Latency SLOs with Histograms

+ Full flexibility in choosing thresholds
+ Full flexibility in choosing aggregation intervals and levels
+ Cost effective (300b / Histogram value)
- Need HDR Histogram Instrumentation
- Need HDR Histogram Metric Store

You can do this with: Circonus, IronDB + Graphite / Grafana, Google internal tooling.

@heinrichhartmann
There are alternative “Mergeable Quantile Summaries” in the literature. Available practical options include:

- circllhist -- Schlossnagle @ Circonus 2013
- HDR Histograms -- Tene @ Azul Systems 2015
- t-digest -- Dunning @ MapR, Erl @ Dynatrace 2015
- DD-Sketch (Histogram) -- Masson @ DataDog 2019
<table>
<thead>
<tr>
<th>Aggregation Method</th>
<th>Accuracy</th>
<th>Performance(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw</td>
<td>35.774 (−)</td>
<td>− / − / −</td>
</tr>
<tr>
<td>circllhist</td>
<td>35.773 (0.001%)</td>
<td>0.86 / 0.000262 / 0.000005</td>
</tr>
<tr>
<td>HDR Histogram</td>
<td>35.775 (0.000%)</td>
<td>3.59 / 0.003... / 0.003...</td>
</tr>
<tr>
<td>t-digest</td>
<td>35.803 (0.029%)</td>
<td>97.00 / 1.900... / 0.003...</td>
</tr>
<tr>
<td>DD-sketch</td>
<td>35.519 (0.256%)</td>
<td>2.39 / 0.003... / 0.000037</td>
</tr>
</tbody>
</table>

(*) This is a quick benchmark of the most popular Python libraries, performed on a 2015 MBP Notebook at [https://github.com/HeinrichHartmann/Statistics-for-Engineers](https://github.com/HeinrichHartmann/Statistics-for-Engineers/)
Conclusion

Percentile Metrics are not suitable for implementing Latency SLOs.

Histogram Metrics allow you to easily calculate arbitrary Latency SLOs.

If you don’t have Histogram metrics, you can do the following:
1. Use log data available over the last days to determine sensible latency thresholds for your service.
2. Add counter metrics for the thresholds
3. Aggregate counter metrics as needed