Latency SLOs Done Right

By Fred Moyer

SCaLE17x
Latency

Is it important?
For any of your services, how many requests were served within 500 ms over the last month?
Latency

For any of your services, how many requests were served within 250ms over the last month?
Latency

How would you answer that question for your services?
Latency

How accurate would your answer be?
I’m Fred and I like SLOs

- Developer Evangelist @Circonus
- Engineer who talks to people
- Writing code and breaking prod for 20 years
- @phredmoyer on Twitter
- Likes C, Go, Perl, PostgreSQL
Talk Agenda

- SLO Refresher
- A Common Mistake
- Computing SLOs with log data
- Computing SLOs by counting requests
- Computing SLOs with histograms
Service Level Objectives

SLI - Service Level Indicator

SLO - Service Level Objectives

SLA - Service Level Agreement
Service Level Objectives

Seeking SRE

The Site Reliability Workbook

Curated and edited by
David N. Blank-Edelman

Edited by Betsy Beyer,
Niall Richard Murphy, David K. Rensin,
Kent Kawahara & Stephen Thorne
SLI - Service Level Indicator, a measure of the service that can be quantified

“99th percentile latency of homepage requests over the past 5 minutes < 300ms”

Excerpted from “SLIs, SLOs, SLAs, oh my!”
@sethvargo @lizthegrey
https://youtu.be/tEylFyxbDL
“SLIs drive SLOs which inform SLAs”

SLO - Service Level Objective, a target for Service Level Indicators

“99th percentile homepage SLI will succeed 99.9% over trailing year”

Excerpted from “SLIs, SLOs, SLAs, oh my!”
@sethvargo @lizthegrey
https://youtu.be/tEylFyxblDE

@phredmoye #SCaLE17x
“SLIs drive SLOs which inform SLAs”

SLA - Service Level Agreement, a legal agreement

“99th percentile homepage SLI will succeed 99% over trailing year”

Excerpted from “SLIs, SLOs, SLAs, oh my!”
@sethvargo @lizthegrey
https://youtu.be/tEylFyxbDL E
Talk Agenda

- SLO Refresher
  - A Common Mistake
  - Computing SLOs with log data
  - Computing SLOs by counting requests
  - Computing SLOs with histograms
A Common Mistake

Averaging Percentiles

\[ p_{95}(W_1 \cup W_2) \neq \frac{(p_{95}(W_1) + p_{95}(W_2))}{2} \]

Works fine when node workload is symmetric

Hides problems when workloads are asymmetric
A Common Mistake
A Common Mistake

99% of requests served here
A Common Mistake
A Common Mistake

\[ p95(W_1) = 220\text{ms} \]
\[ p95(W_2) = 650\text{ms} \]

\[ p95(W_1 \cup W_2) = 230\text{ms} \]

\[ \frac{p95(W_1)+p95(W_2)}{2} = 430\text{ms} \]

\~200\%\ difference

@phredmoye #SCaLE17x
A Common Mistake

- p95 actual (230ms)
- p95 average (430ms)
- ERROR
A Common Mistake

Log parser => Metrics (mtail)

What metrics are you storing?

Averages? p50, p90, p95, p99, p99.9, p99.9?
Talk Agenda

- SLO Refresher
- A Common Mistake
  - Computing SLOs with log data
  - Computing SLOs by counting requests
  - Computing SLOs with histograms
Computing SLOs with log data

"%%d/%b/%Y %T}\%{msec_frac}\%{%z}\%

~100 bytes per log line

~1GB for 10M requests
Computing SLOs with log data

Logs => HDFS

Logs => ElasticSearch/Splunk

ssh -- `grep ... | awk ... > 550 ... | wc -l`

Then query all the log files
Computing SLOs with log data

Calculating p95 SLI

1. Extract samples for time window
2. Sort the samples by value
3. Find the sample 5% count from largest
4. That’s your p95
Computing SLOs with log data

Calculating p95 SLO

“95th percentile SLI will succeed 99.9% trailing year”

1. Divide 1 year samples into 1,000 slices
2. For each slice, calculate SLI
3. Was p95 SLI met for 999 slices? Met SLO if so
Computing SLOs with log data

Pros:

1. Easy to configure logs to capture latency
2. Easy to roll your own processing code, some open source options out there
3. Accurate results
Computing SLOs with log data

Cons:

1. Expensive (see log analysis solution pricing)
2. Sampling possible but skews accuracy
3. Slow
4. Difficult to scale
Talk Agenda

- SLO Refresher
- A Common Mistake
- Computing SLOs with log data
  - Computing SLOs by counting requests
  - Computing SLOs with histograms
Computing SLOs by counting requests

1. Count # of requests that violate SLI threshold
2. Count total number of requests
3. % success = 100 - (#failed_reqs/#total_reqs)*100

Similar to Prometheus cumulative ‘le’ histogram
Computing SLOs by counting requests

@phredmoye

#SCaLE17x
Computing SLOs by counting requests

SLO = 90% of reqs < 30ms

# bad requests = 2,262
# total requests = 60,124

100 - \( \frac{2262}{60124} \) * 100 = 96.2%

SLO was met
Computing SLOs by counting requests

**Pros:**

1. Simple to implement
2. Performant
3. Scalable
4. Accurate
Computing SLOs by counting requests

Cons:

1. Fixed SLO threshold - must reconfigure
2. Look back impossible for other thresholds
Talk Agenda

- SLO Refresher
- A Common Mistake
- Computing SLOs with log data
- Computing SLOs by counting requests
  - Computing SLOs with histograms
Computing SLOs with histograms

AKA distributions

Sample counts in bins/buckets

Gil Tene’s hdrhistogram.org

# Samples

Sample value

Mode

Median

q(0.9)

q(0.5)

Mean

q(1)

#SCaLE17x
Computing SLOs by counting requests

Some histogram types:

1. Linear
2. Approximate
3. Fixed bin
4. Cumulative
5. Log Linear
Log Linear Histogram

github.com/circonus-labs/libcircllhist
github.com/circonus-labs/circonusllhist

@phredmoye

#SCaLE17x
Log Linear Histogram
Mergeability

\[ h(A \cup B) = h(A) \cup h(B) \]

A & B must have identical bin boundaries

Can be aggregated both in space and time
How many requests are faster than 330ms?

1. Walk the bins lowest to highest until you reach 330ms
2. Sum the counts in those bins
3. Done
This is brilliant. However worth noting is that you still do have to make sure values you pick are in a histogram bin line. Make sure you know what your binning algorithm is.

Fred Moyer @phredmoyer
Slides from my lightning talk "Latency SLOs done right" at newopdays, hosted at @splunk slideshare.net/redhotpenguin/...
So ... where are the bin boundaries?

For the libcirclllhist implementation we have bins at:

... 320, 330, 340, ...

... And: 10,11,12,13...

... And: 0.0000010, 0.0000011, 0.0000012,

For every decimal floating point number, with 2 significant digits, we have a bin (within $10^{+/128}$).
Computing SLOs with histograms

Pros:
1. Space Efficient (HH: ~ 300bytes / histogram in practice, 10x more efficient than logs)
2. Full Flexibility:
   - Thresholds can be chosen as needed and analyzed
   - Statistical methods applicable, IQR, count_below, q1, etc.
3. Mergability (HH: Aggregate data across nodes)
4. Performance (ns insertions, μs percentile calculations)
5. Bounded error (half the bin size)
6. Several open source libraries available
Computing SLOs with histograms

Cons:

1. Math is more complex than other methods
2. Some loss of accuracy (<<5%) in worst cases
Log Linear histograms with Python

github.com/circonus-labs/libcircllhist
(autoconf && ./configure && make && make install)

github.com/circonus-labs/libcircllhist/tree/master/src/python
(pip install circllhist)
Log Linear histograms with Python

```python
h = Circllhist()  # make a new histogram
h.insert(123)    # insert value 123
h.insert(456)    # insert value 456
h.insert(789)    # insert value 789
print(h.count()) # prints 3
print(h.sum())   # prints 1,368
print(h.quantile(0.5)) # prints 456
```
Log Linear histograms with Python

from matplotlib import pyplot as plt
from circlllhist import Circlllhist
H = Circlllhist()
... # add latency data to H via insert()
H.plot()
plt.axvline(x=H.quantile(0.95), color='red')
Log Linear histograms with Python
Conclusions

1. Averaging Percentiles is tempting, but misleading

2. Use counters or histograms to calculate SLOs correctly

3. Histograms give the most flexibility in choosing latency thresholds, but only a couple libraries implement them (libcircllhist, hdrhistogram)

4. Full support for (sparsely encoded-, HDR-) histograms in TSDBs still lacking (except IRONdb).
Thank you!

Tweet me: @phredmoyer

AMA about histograms on: slack.s.circonus.com

More talks about histograms:

slideshare.net/redhotpenguin

https://github.com/HeinrichHartmann/DS4OPS

@phredmoye

#SCaLE17x
DEMO