Applying Statistics to Root-Cause Analysis

Karthik Kumar

Intros

Me:

- Software engineer, building root-cause analysis tools
- Interested in software performance and reliability

Lightstep:

- Simple Observability for Deep Systems
- Distributed tracing focused (CEO/co-founder created Dapper)

Topics

Data Complexity

- Distributions
- Correlations

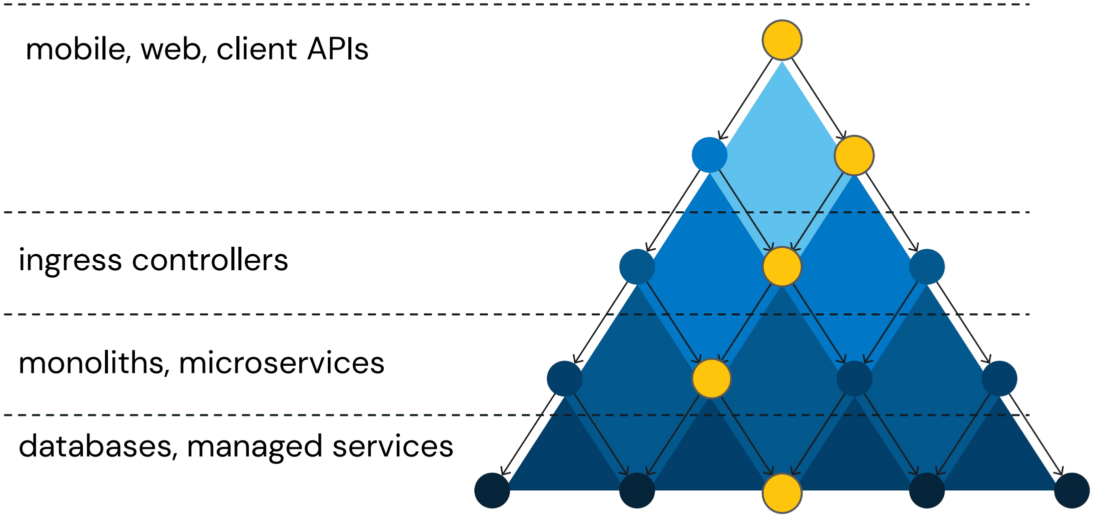
Data Quantity

- Bias
- Sampling

“The function of good software is to make the complex appear to be simple.”

– Grady Booch; ACM Fellow, creator of UML

System & Telemetry Complexity



Traces provide rich, contextual data but root-cause analysis can be difficult and expensive

Maximizing insights & minimizing complexity

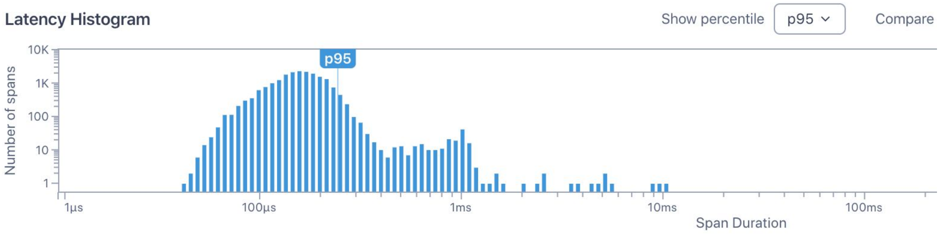
Distributions

1. Model performance as a shape, not a number (histograms, not averages)
2. Visually compare changes in performance

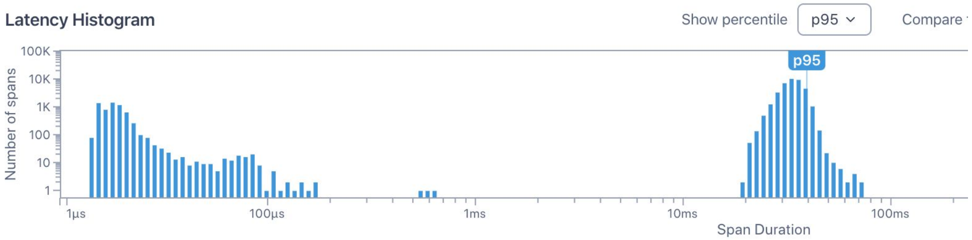
Correlations

Modeling common behaviors with latency distributions

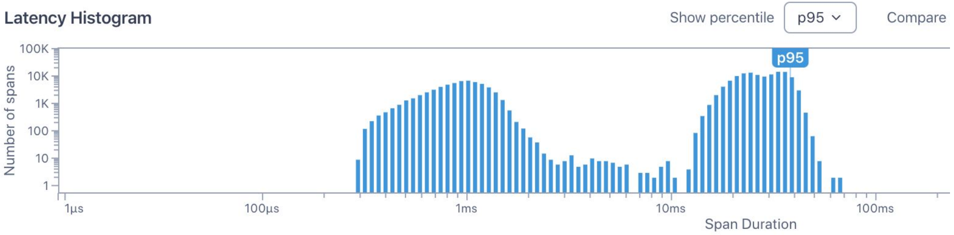
Long tail latency



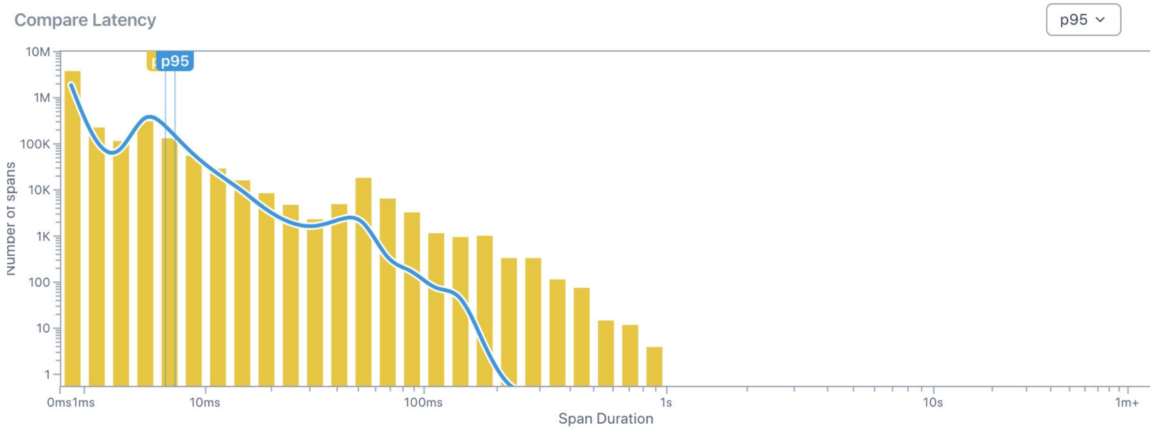
Cache hit / error path



Different classes of requests



Comparing Distributions



Before a deployment

After a deployment

Maximizing insights & minimizing complexity

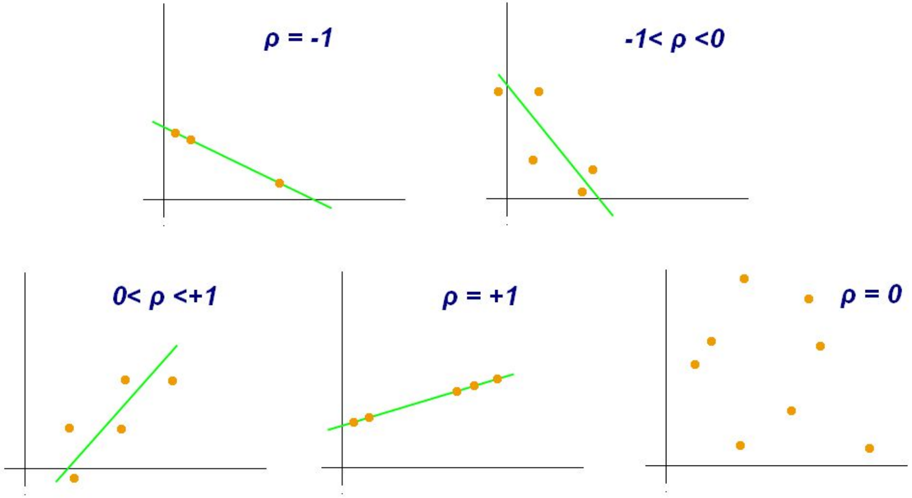
Distributions

Correlations

1. Associate specific **behaviors** of different **subpopulations**
    a. Behaviors: latency, errors (Y)
    b. Subpopulations: spans with tag, service/operation on critical path (X)
2. *Automatically* identify subpopulations with sufficient correlation
3. Present information in understandable way

Correlations

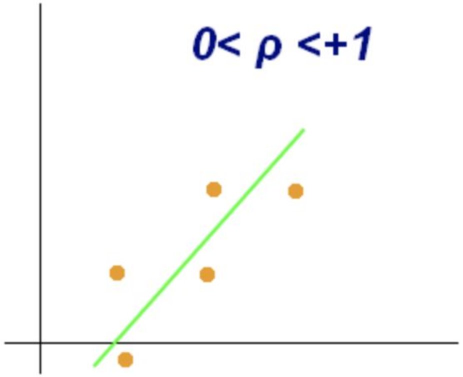
Pearson Correlation Coefficient: simple linear correlation between two (potentially binary) variables X, Y



Positively correlated with latency

Statistics

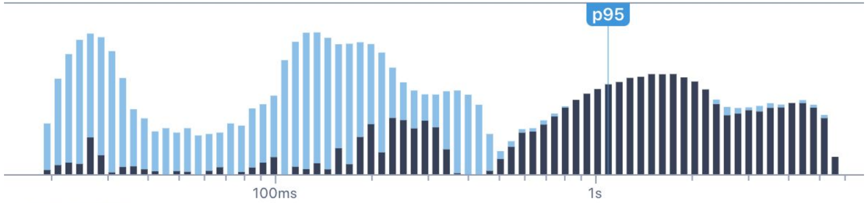
Latency
of spans
in sample



Spans with tag
“`status: payment succeeded`”

Root-cause analysis

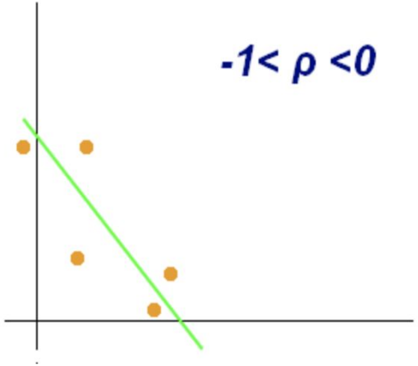
status: payment succeeded **+.72**
On tag in trace



Negatively correlated with latency

Statistics

Latency of spans in sample

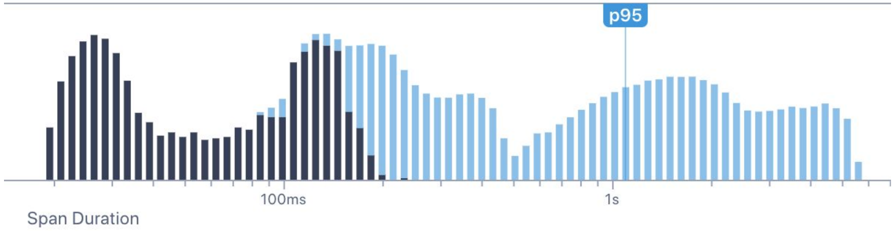


Spans with tag “http.method: GET”

Root-cause analysis

http.method: GET -.80

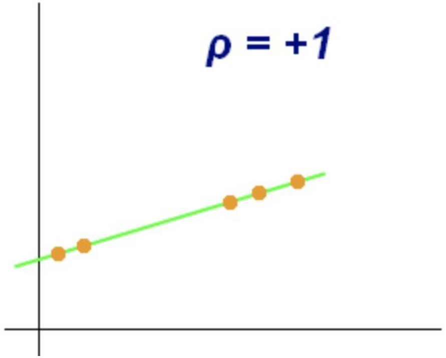
On tag in result



Perfectly correlated with errors

Statistics

Spans with errors in sample



Spans with tag
"`http.status_code: 400`"

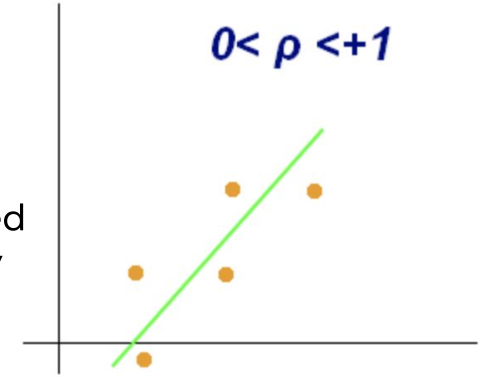
Root-cause analysis

http.status_code: 400 **+1.00**
On tag in result

Positively correlated with user-specified behaviors

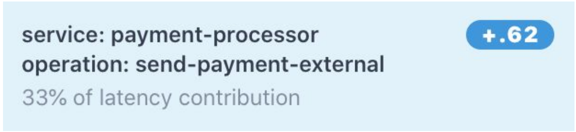
Statistics

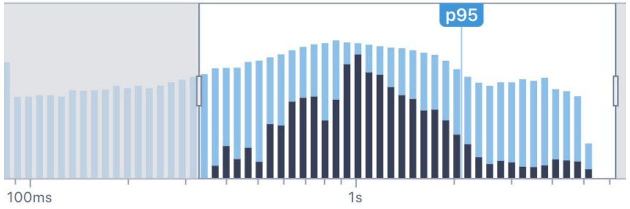
Spans inside selected latency region



Spans with
“service: payment-processor”
“operation: send-payment-external”

Root-cause analysis





It really works!

Thomas Millar
@thmsmlr

Debugged a production issue with @LightStepHQ new correlation feature in seconds. Really makes me wonder what's possible with distributed tracing when you start reasoning about the traces in aggregate.

5:49 PM · Mar 11, 2019 · Twitter Web Client

Alex Hidalgc
@ahidalgosre

Replying to @Di4naO @lizthegrey and 5 others

Just the other day LightStep automatically surfaced a problem for us. The correlation engine basically said, "Hey, this thing is normally like _this_ but right now it's suddenly like _this_?" The responding Engineer estimates it took 1/10th the time it would have otherwise.

1:51 PM · Sep 4, 2019 · Twitter Web App

Pearson Correlation Coefficient Pros/Cons

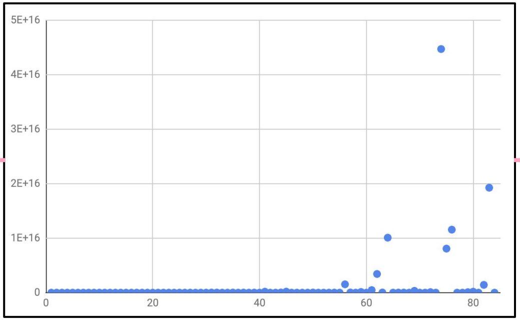
- \+ Unit of measurement does not affect calculation
- \+ Simple to understand and implement
- \+ Works well for most cases

- \- Only measures linear association between X & Y
- \- Possibility of Type 1 and Type 2 errors, since dataset is a sample of the population

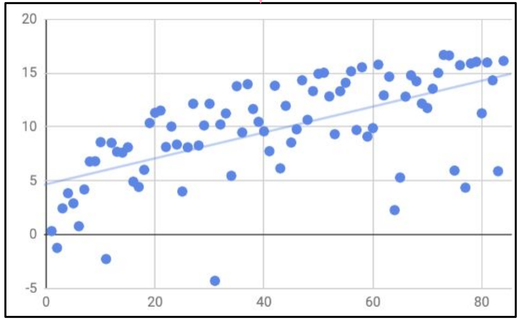
$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

For a population (covariance of X, Y divided by the product of standard deviation)

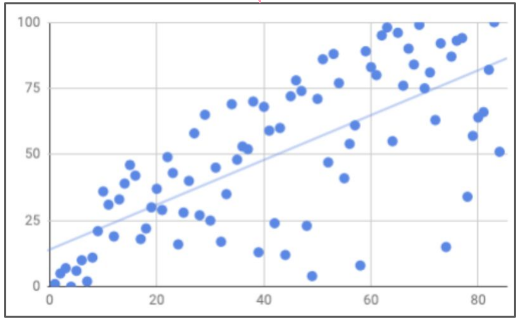
Transformation of non-linear datasets



Highly skewed distribution



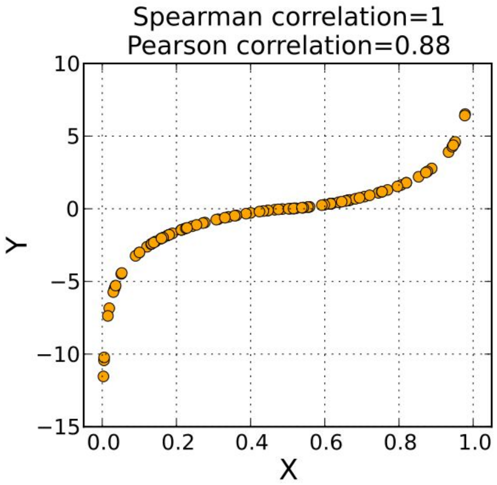
Log Transformation



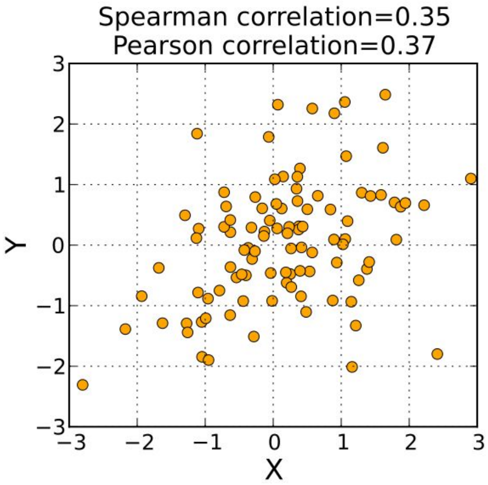
Percentile Rank Transformation

Correlation on non-linear datasets

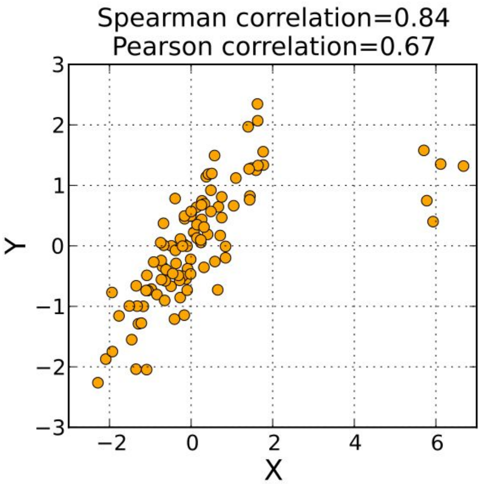
- Use Spearman's Rank Correlation Coefficient
  - Measures how well the relationship between **the rankings of two variables** can be described using a monotonic function.



Non-linear increasing function modeled well by Spearman's



Similar results without outliers



Spearman's more resistant to outliers

https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

Correlating with more properties

- Since a “subpopulation” is just a feature of traces, we can correlate latency and errors with other properties:
  - Call patterns (serial, scatter-gather etc)
  - Logs on spans
  - Existence of certain spans up/down the trace

Takeaways

- Tracing data is noisy and complex
- Use histograms to model system performance
- Use simple statistical analysis to expose patterns, guide hypothesis validation and optimize root-cause analysis with traces

Topics

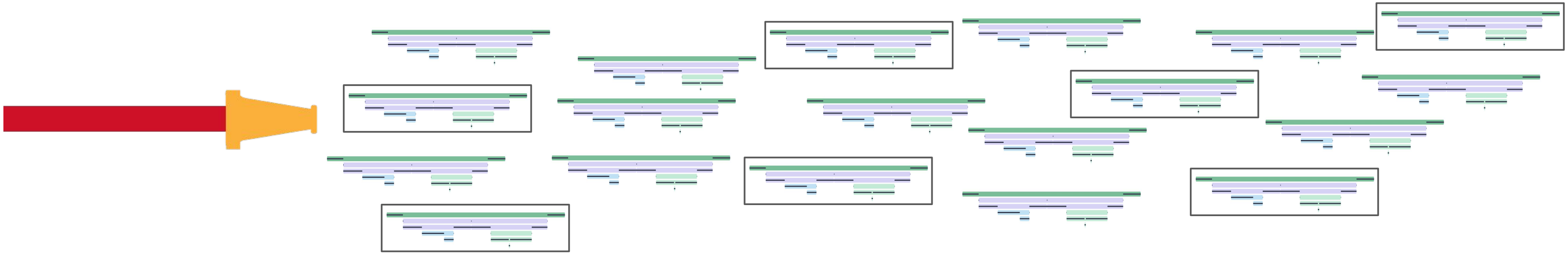
Data Complexity

- Distributions
- Correlations

Data Quantity

- Bias
- Sampling

What data is relevant to the user?



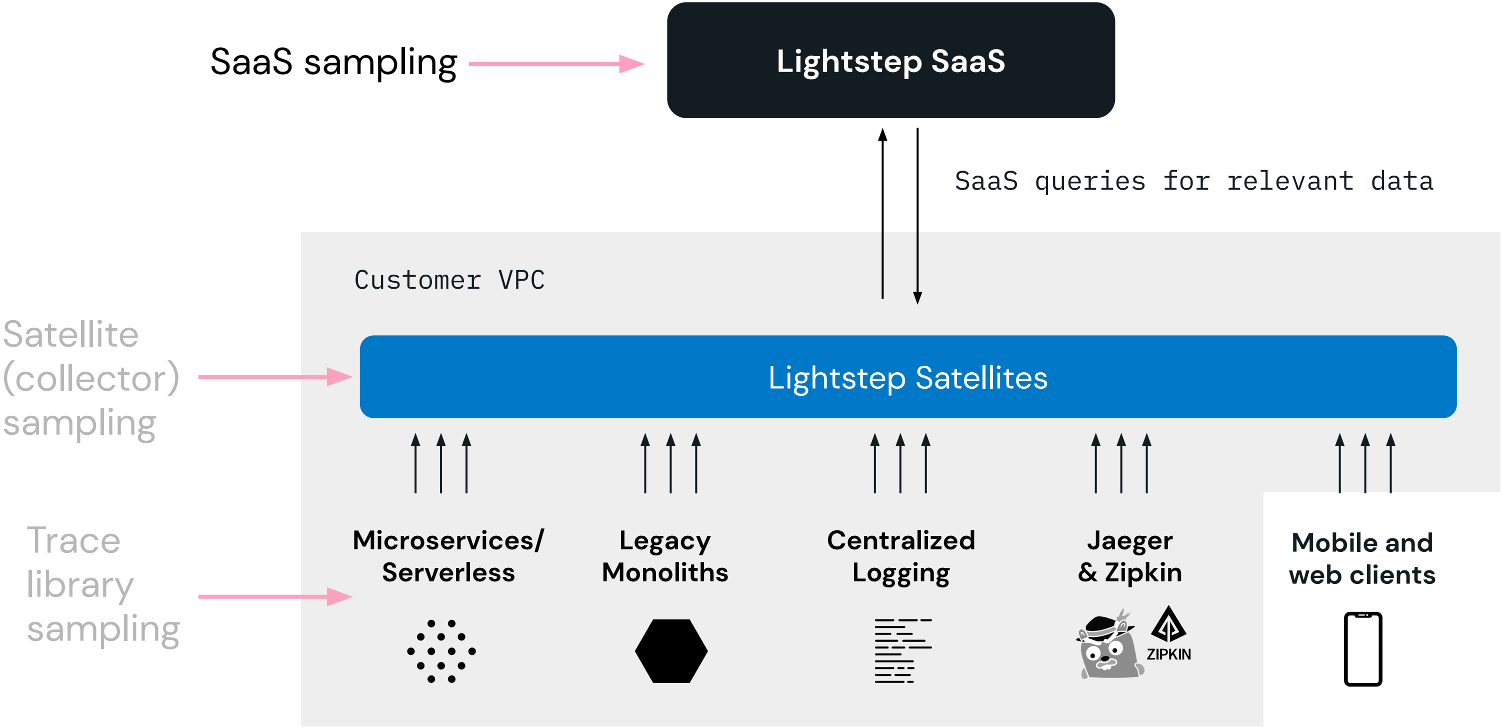
Goal: Focus our sampling budget on interesting traces

- Anything user cares about (real-time or saved)
- Ingress operations
  - Constant stream of data being collected in the background for each service's (entry-point) operations (to support SLA reporting)

Why is bias important?

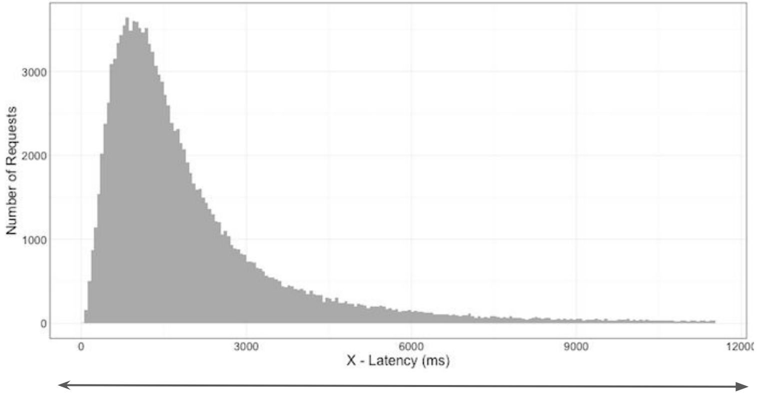
- We want to guide humans to root-causes.
- It is possible to automatically identify subpopulations of interest
- Goal with sampling:
    - Capture “some” or “enough” traces for as many different interesting subpopulations as possible. It isn’t useful if the majority of our post-sampled data reflects the normal-case behavior.

Tracing Architecture



How do we bias the sampling?

- Sample error traces
- Sample traces across latency range
  - To bias towards capturing tail behavior, better than uniform sampling



Equally likely to be sampled

https://blog.newrelic.com/wp-content/uploads/right-skewed-long-tail-distribution.png

Sampling Requirements (at the SaaS)

- Input: stream of traces of unknown length
- Output: a representative sample. Use the sample to try to answer questions about the original population as a whole
- Efficient sampling decisions
- Works in a distributed setting (without centralized coordination)

VarOpt Sampling (2010)

Stream sampling for variance-optimal estimation of subset sums*

Edith Cohen[†] Nick Duffield[†] Haim Kaplan[‡] Carsten Lund[†] Mikkel Thorup[†]

Abstract

From a high volume stream of weighted items, we want to maintain a generic sample of a certain limited size k that we can later use to estimate the total weight of arbitrary subsets. This is the classic context of on-line reservoir sampling, thinking of the generic sample as a reservoir. We present an efficient reservoir sampling scheme, VarOpt_k, that dominates all previous schemes in terms of estimation quality. VarOpt_k provides *variance optimal unbiased estimation of subset sums*. More precisely, if we have seen n items of the stream, then for *any* subset size m, our scheme based on k samples minimizes the average variance over all subsets of size m. In fact, the optimality is against any off-line scheme with k samples tailored for the concrete set of items seen. In addition to optimal average variance, our scheme provides tighter worst-case bounds on the variance of *particular* subsets than previously possible. It is efficient, handling each new item of the stream in $O(\log k)$ time. Finally, it is particularly well suited for combination of samples from different streams in a distributed setting.

https://arxiv.org/pdf/0803.0473.pdf

VarOpt Sampling

- Online reservoir sampling scheme
- Each item in input sequence has an attached weight ("importance").
- Produces an adjusted weight (different from the input weight) for each sampled item

VarOpt Meets Our Requirements

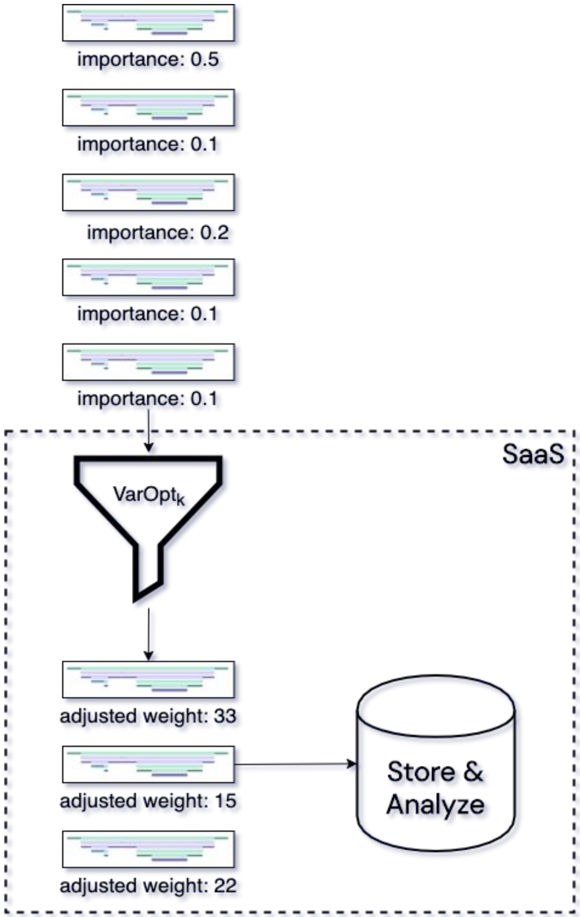
- Minimizes average variance over subsets
  - Subset-sum weights can be used to answer quantile queries (what percentile is this trace in the population?)
- Efficient sampling decisions - O(log k)
- Works in a distributed setting - generalized recurrence

VarOpt Sampling

A high volume stream of ***n*** traces for each ingress operation.
Assign importances (weights) to bias towards “interesting” traces.

Sample ***k*** items that minimizes average variance of arbitrary subsets in *O****(n log k)*** time.

Use subsets of traces and adjusted weights to calculate quantile measurements (for Correlations, aggregate critical path).



Takeaways

- Tracing is data intensive, but not all data is worth analyzing
- We have several opportunities for sampling and each has different constraints and requirements
- We want to bias towards storing and analyzing “interesting” traces and we should be flexible in defining “interesting”-ness
- For sampling on the Saas-side, one option that worked for us is VarOpt.

Summary

Data Complexity

- Maximizing insights, minimizing complexity
- Distributions, Correlations

Tracing Data Quantity

- Maximizing relevance, minimizing cost
- Bias, Sampling

Questions?

karthik@lightstep.com
@karkum

Extra Slides

Maximizing insights & minimizing complexity

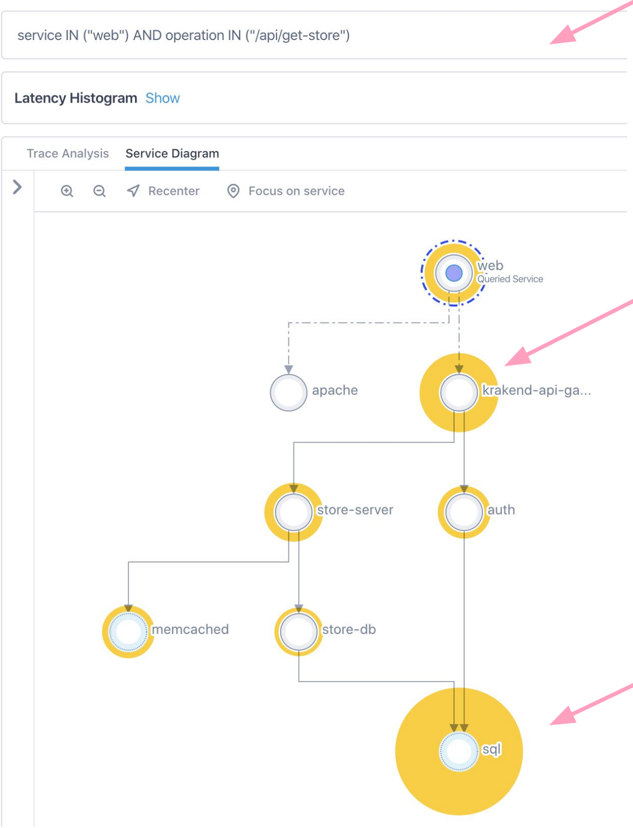
Distributions

Correlations

Dynamic system diagrams

1. Gather a population of traces filtered by a certain condition
    a. Ex: service=”api” && operation=”create-user” && tag=”host:abc”
2. Identify and aggregate critical path
3. Preserve hierarchy and draw a diagram

Latency Service Diagrams



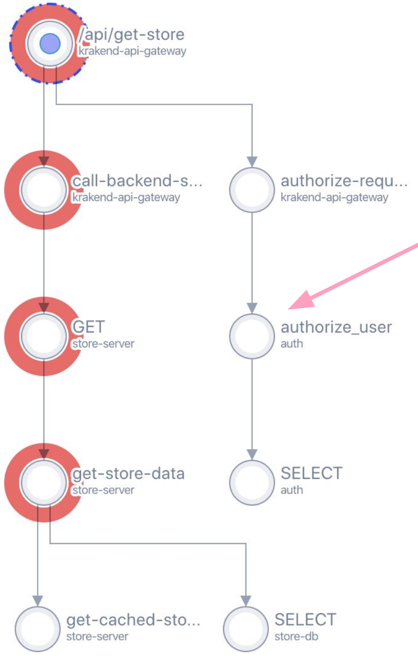
Define traces of interest

Highlight aggregate critical path of request

Inferred through client spans; not explicitly traced

Error Operation Diagrams

Highlights *operations* with errors



“Innocent” (non-error) operations