Prometheus Histograms – Past, Present, and Future

Björn “Beorn” Rabenstein
PromCon EU, Munich – 2019-11-08

Grafana Labs

This is not a Howto.

Visit https://prometheus.io/docs/practices/histograms/ instead…

The Past

~~The Past~~

The Present

The Present

Part 1: What works really well

Mathematically correct aggregation.

By Apdex - Apdex Web site, Fair use,
https://en.wikipedia.org/w/index.php?curid=8994240

High frequency sampling feasible.

“What percentage of requests in the last hour got a response in 100ms or less?”

“How many HTTP responses larger than 4kiB were served on 2019-11-03 between 02:30 and 02:45?”

Mathematically correct aggregation. *

*

By Apdex - Apdex Web site, Fair use,
https://en.wikipedia.org/w/index.php?curid=8994240

High frequency sampling feasible.

“What percentage of requests in the last hour got a response in 100ms or less?” *

“How many HTTP responses larger than 4kiB were served on 2019-11-03 between 02:30 and 02:45?” *

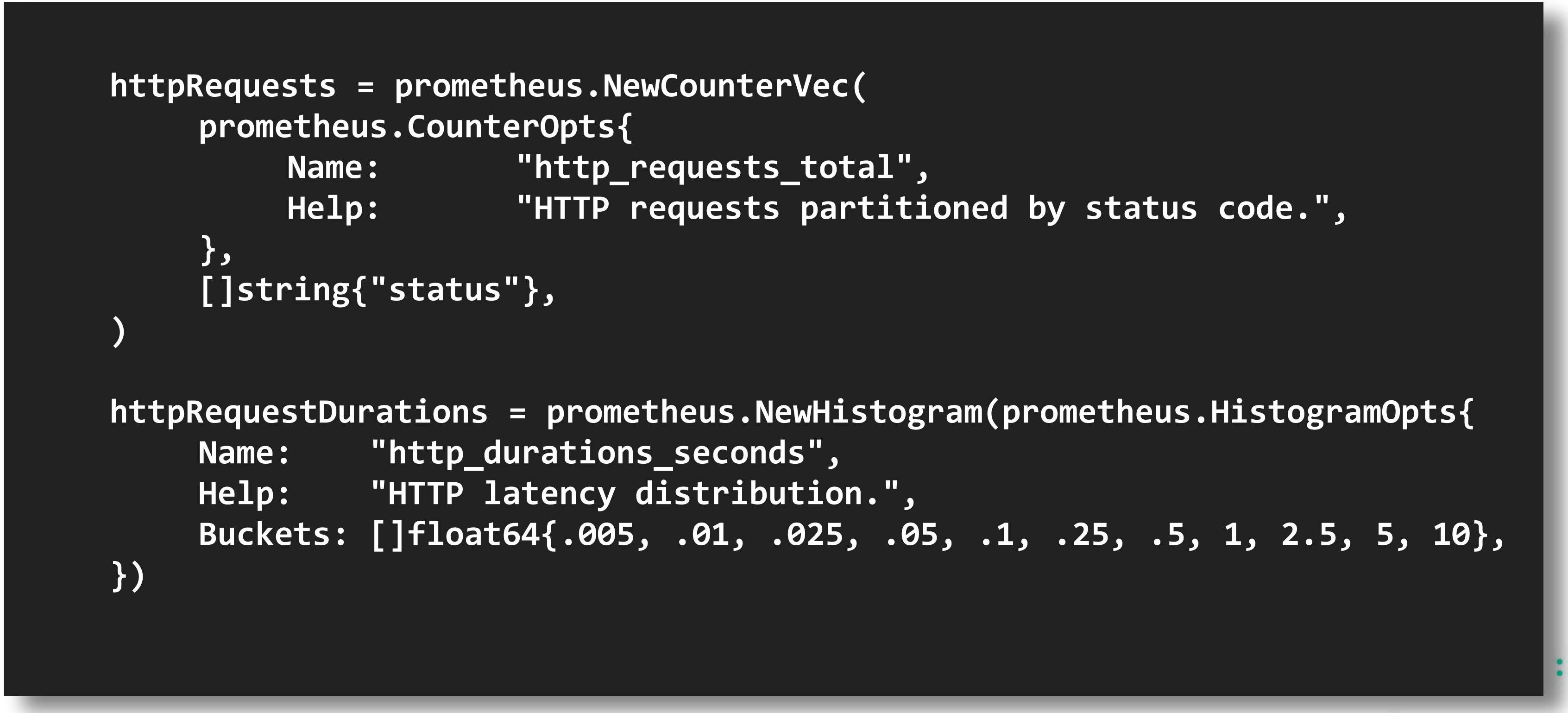
* If suitable buckets defined.

The Present

Part 2: An incomplete list of problems

```
histogram_quantile(0.99, sum(rate(rpc_duration_seconds_bucket[5m])) by (le))
histogram_quantile(0.99, sum(rate(rpc_duration_seconds_bucket[5m])) by (le))
```

- Accuracy depends on bucket layout.
- Bucketing scheme must be compatible…
  - …across the aggregated metrics.
  - …across the range of the rate calculation.
- Lack of ingestion isolation can wreak havoc.



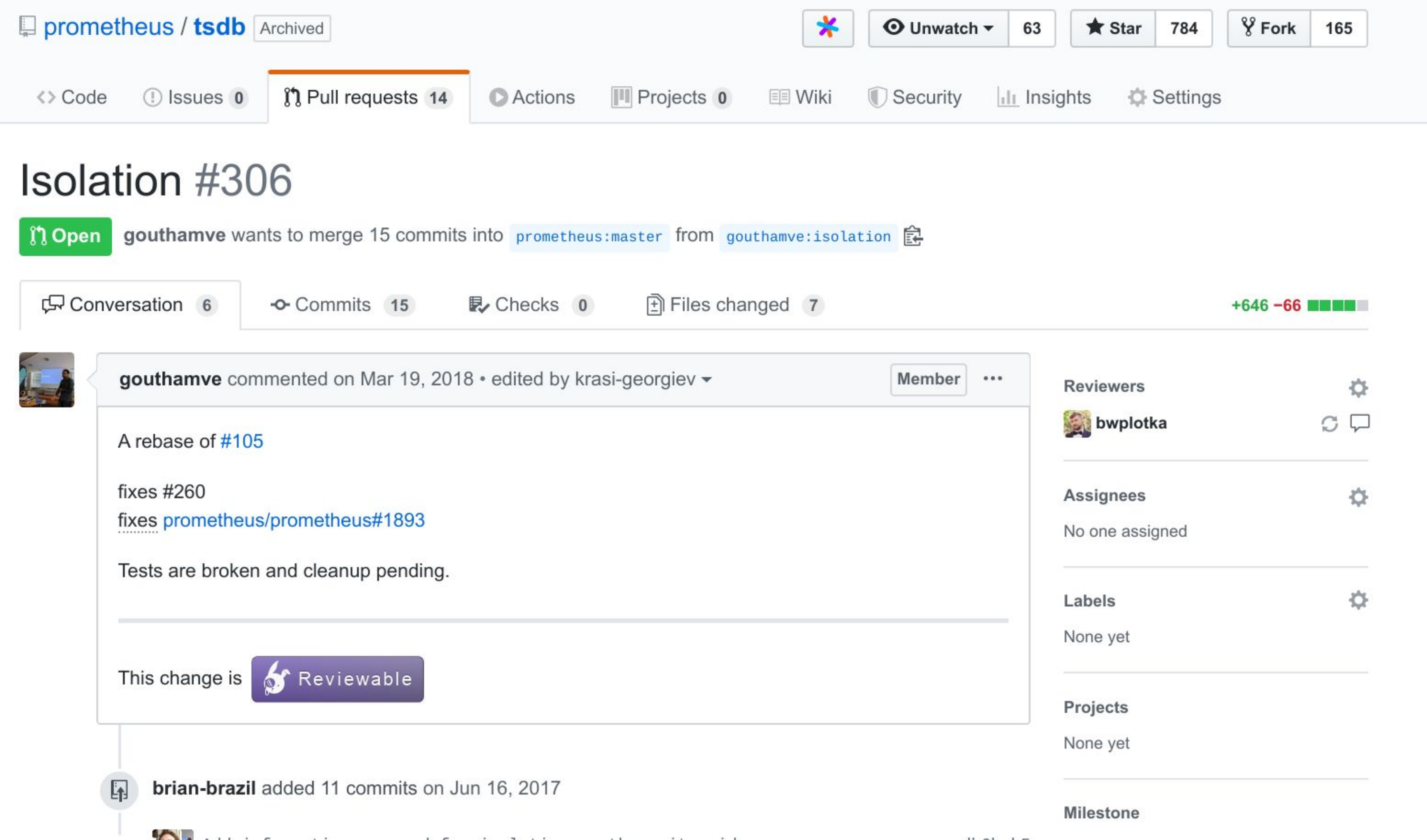
```
httpRequests = prometheus.NewCounterVec(
    prometheus.CounterOpts{
        Name:       "http_requests_total",
        Help:       "HTTP requests partitioned by status code.",
    },
    []string{"status"},
)

httpRequestDurations = prometheus.NewHistogram(prometheus.HistogramOpts{
    Name:    "http_durations_seconds",
    Help:    "HTTP latency distribution.",
    Buckets: []float64{.005, .01, .025, .05, .1, .25, .5, 1, 2.5, 5, 10},
})
```

The Future

The Future

Option 0: Fix isolation.



Isolation #306

gouthamve wants to merge 15 commits into `prometheus:master` from `gouthamve:isolation`

gouthamve commented on Mar 19, 2018 • edited by krasi-georgiev

A rebase of #105

fixes #260
fixes prometheus/prometheus#1893

Tests are broken and cleanup pending.

This change is

brian-brazil added 11 commits on Jun 16, 2017

The Future

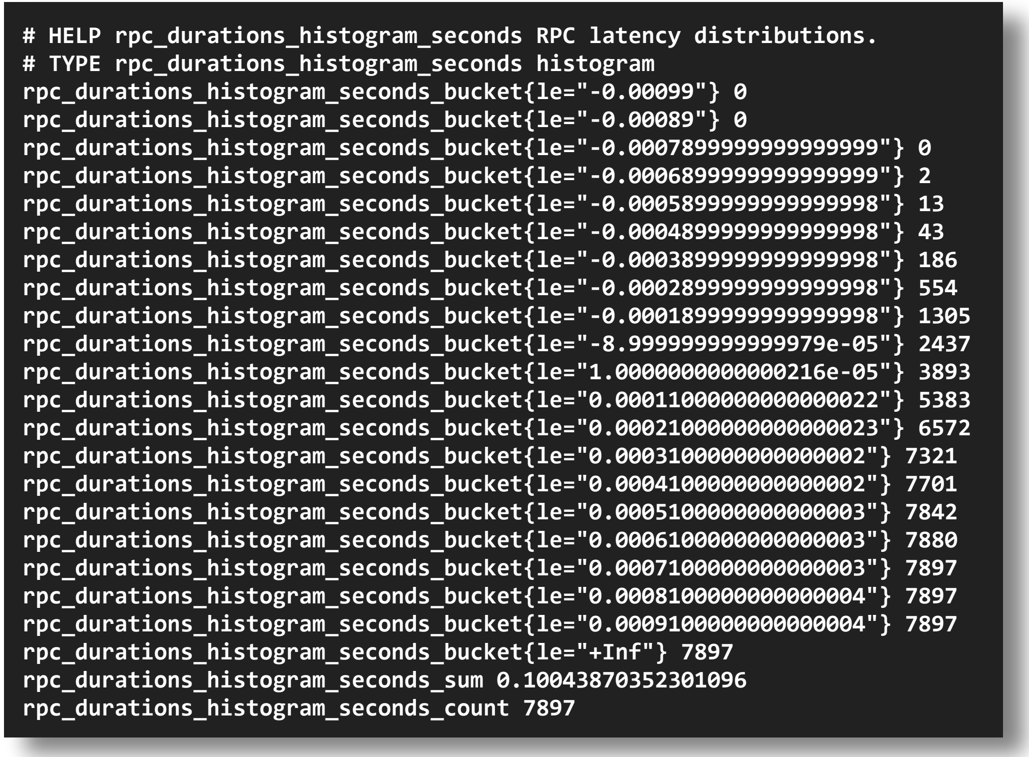
Option 1: Do nothing.

Instrument first,
ask questions later.

The Future

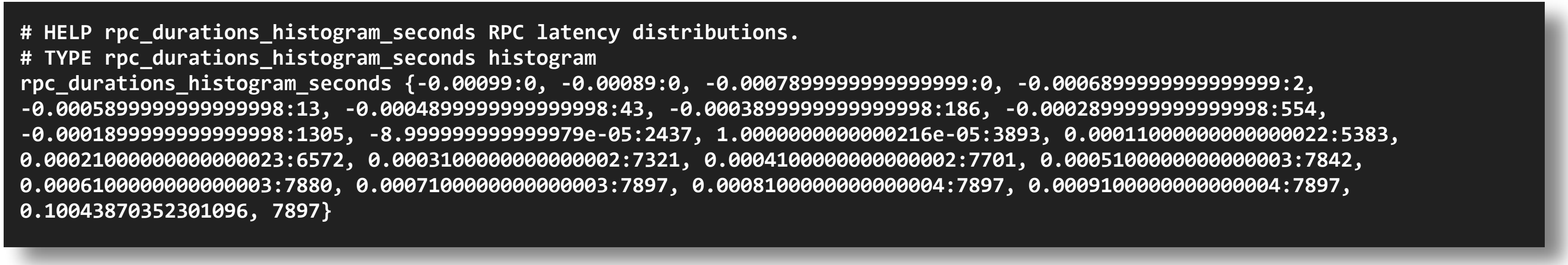
Option 2: Make buckets a bit cheaper.

Option 2a: Change exposition format



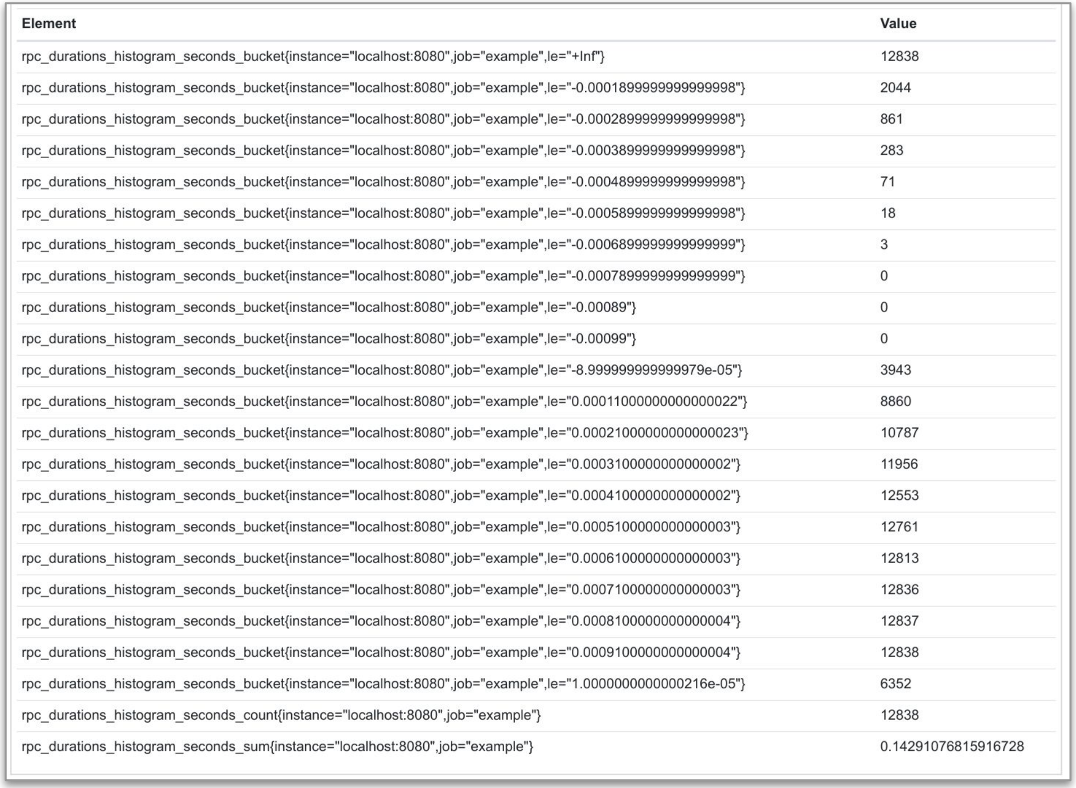
```
# HELP rpc_durations_histogram_seconds RPC latency distributions.
# TYPE rpc_durations_histogram_seconds histogram
rpc_durations_histogram_seconds_bucket{le="-0.00099"} 0
rpc_durations_histogram_seconds_bucket{le="-0.00089"} 0
rpc_durations_histogram_seconds_bucket{le="-0.0007899999999999999"} 0
rpc_durations_histogram_seconds_bucket{le="-0.0006899999999999999"} 2
rpc_durations_histogram_seconds_bucket{le="-0.0005899999999999998"} 13
rpc_durations_histogram_seconds_bucket{le="-0.0004899999999999998"} 43
rpc_durations_histogram_seconds_bucket{le="-0.0003899999999999998"} 186
rpc_durations_histogram_seconds_bucket{le="-0.0002899999999999998"} 554
rpc_durations_histogram_seconds_bucket{le="-0.0001899999999999998"} 1305
rpc_durations_histogram_seconds_bucket{le="-8.999999999999979e-05"} 2437
rpc_durations_histogram_seconds_bucket{le="1.0000000000000216e-05"} 3893
rpc_durations_histogram_seconds_bucket{le="0.00011000000000000022"} 5383
rpc_durations_histogram_seconds_bucket{le="0.00021000000000000023"} 6572
rpc_durations_histogram_seconds_bucket{le="0.0003100000000000002"} 7321
rpc_durations_histogram_seconds_bucket{le="0.0004100000000000002"} 7701
rpc_durations_histogram_seconds_bucket{le="0.0005100000000000003"} 7842
rpc_durations_histogram_seconds_bucket{le="0.0006100000000000003"} 7880
rpc_durations_histogram_seconds_bucket{le="0.0007100000000000003"} 7897
rpc_durations_histogram_seconds_bucket{le="0.0008100000000000004"} 7897
rpc_durations_histogram_seconds_bucket{le="0.0009100000000000004"} 7897
rpc_durations_histogram_seconds_bucket{le="+Inf"} 7897
rpc_durations_histogram_seconds_sum 0.10043870352301096
rpc_durations_histogram_seconds_count 7897
```

plaintext	1676 bytes
gzip'd	313 bytes
protobuf	357 bytes
protobuf gzip'd	342 bytes



```
# HELP rpc_durations_histogram_seconds RPC latency distributions.
# TYPE rpc_durations_histogram_seconds histogram
rpc_durations_histogram_seconds {-0.00099:0, -0.00089:0, -0.0007899999999999999:0, -0.0006899999999999999:2,
-0.0005899999999999998:13, -0.0004899999999999998:43, -0.0003899999999999998:186, -0.0002899999999999998:554,
-0.0001899999999999998:1305, -8.999999999999979e-05:2437, 1.0000000000000216e-05:3893, 0.000110000000000000022:5383,
0.000210000000000000023:6572, 0.00031000000000000002:7321, 0.00041000000000000002:7701, 0.00051000000000000003:7842,
0.00061000000000000003:7880, 0.00071000000000000003:7897, 0.00081000000000000004:7897, 0.00091000000000000004:7897,
0.10043870352301096, 7897}
```

Option 2b: Change TSDB



Element	Value
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="+Inf"}	12838
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00018999999999999998"}	2044
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00028999999999999998"}	861
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00038999999999999998"}	283
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00048999999999999998"}	71
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00058999999999999998"}	18
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00068999999999999999"}	3
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00078999999999999999"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00089"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-0.00099"}	0
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="-8.999999999999979e-05"}	3943
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00011000000000000022"}	8860
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00021000000000000023"}	10787
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00031000000000000002"}	11956
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00041000000000000002"}	12553
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00051000000000000003"}	12761
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00061000000000000003"}	12813
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00071000000000000003"}	12836
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00081000000000000004"}	12837
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="0.00091000000000000004"}	12838
rpc_durations_histogram_seconds_bucket{instance="localhost:8080",job="example",le="1.0000000000000216e-05"}	6352
rpc_durations_histogram_seconds_count{instance="localhost:8080",job="example"}	12838
rpc_durations_histogram_seconds_sum{instance="localhost:8080",job="example"}	0.14291076815916728

The Future

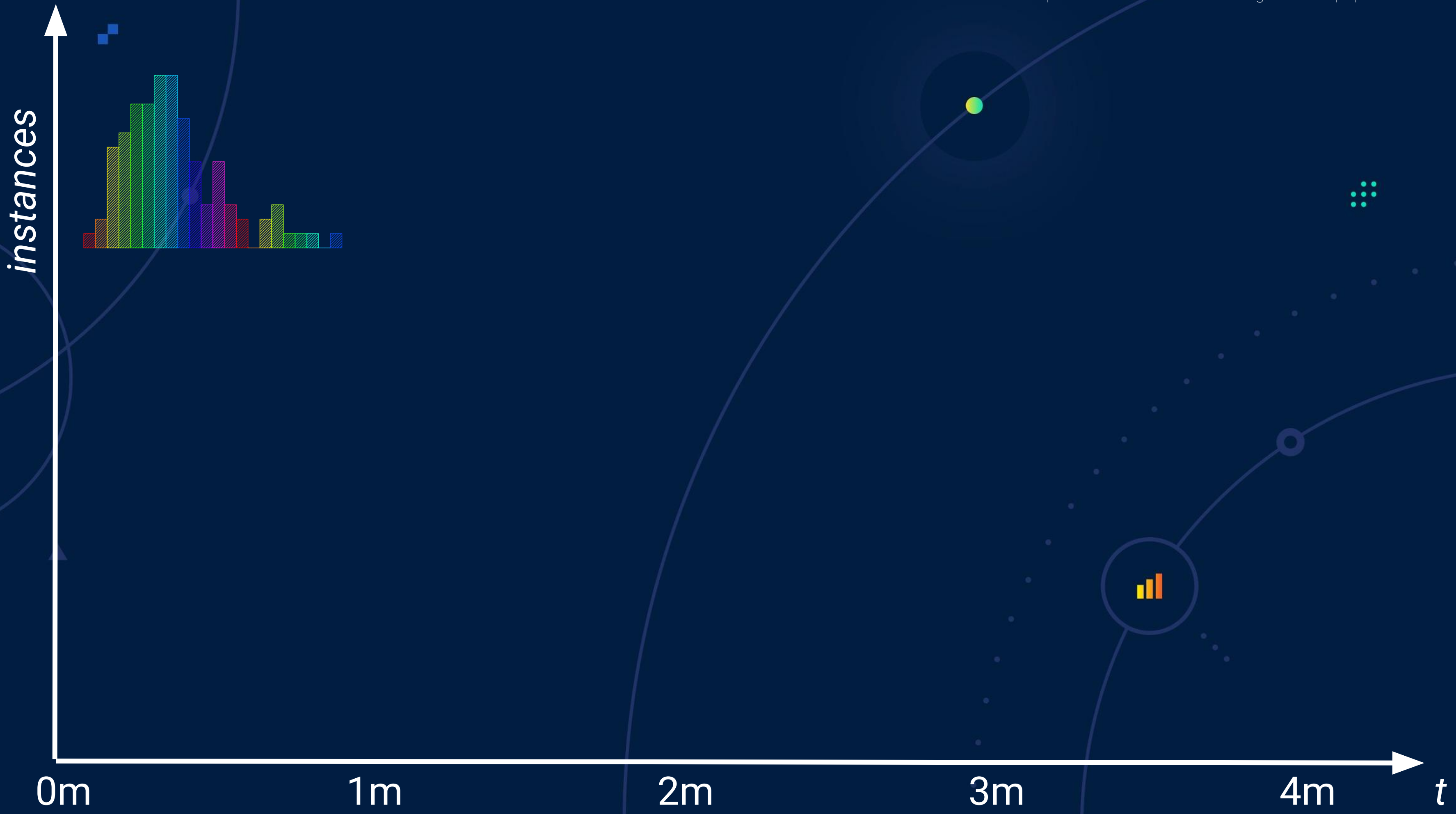
Option 3: Make buckets a lot cheaper.

HdrHistogram: http://hdrhistogram.org

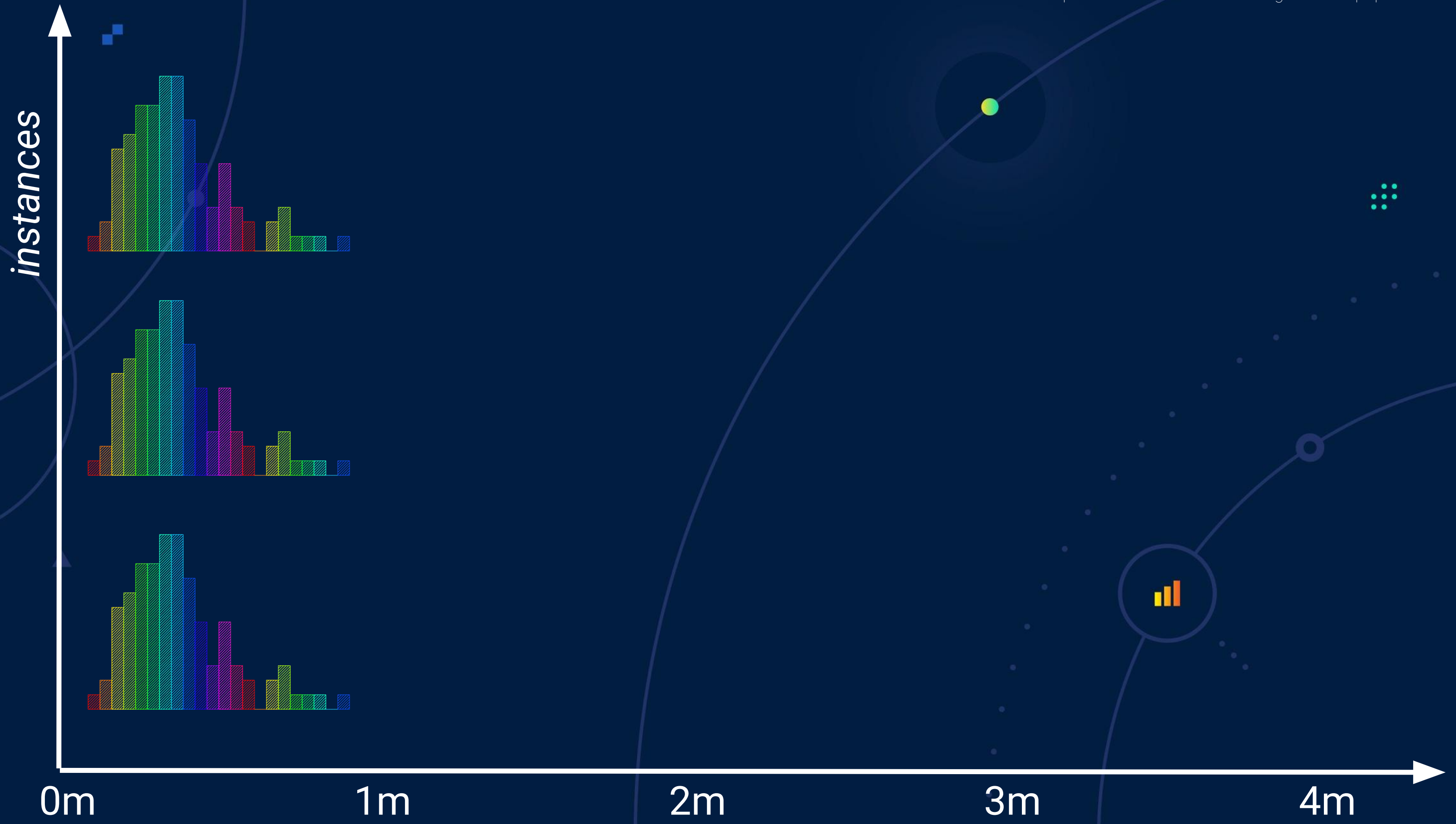
Circonus’s Circllhist: https://github.com/circonus-labs/libcircllhist/

Datadog’s DDSketch: https://arxiv.org/abs/1908.10693

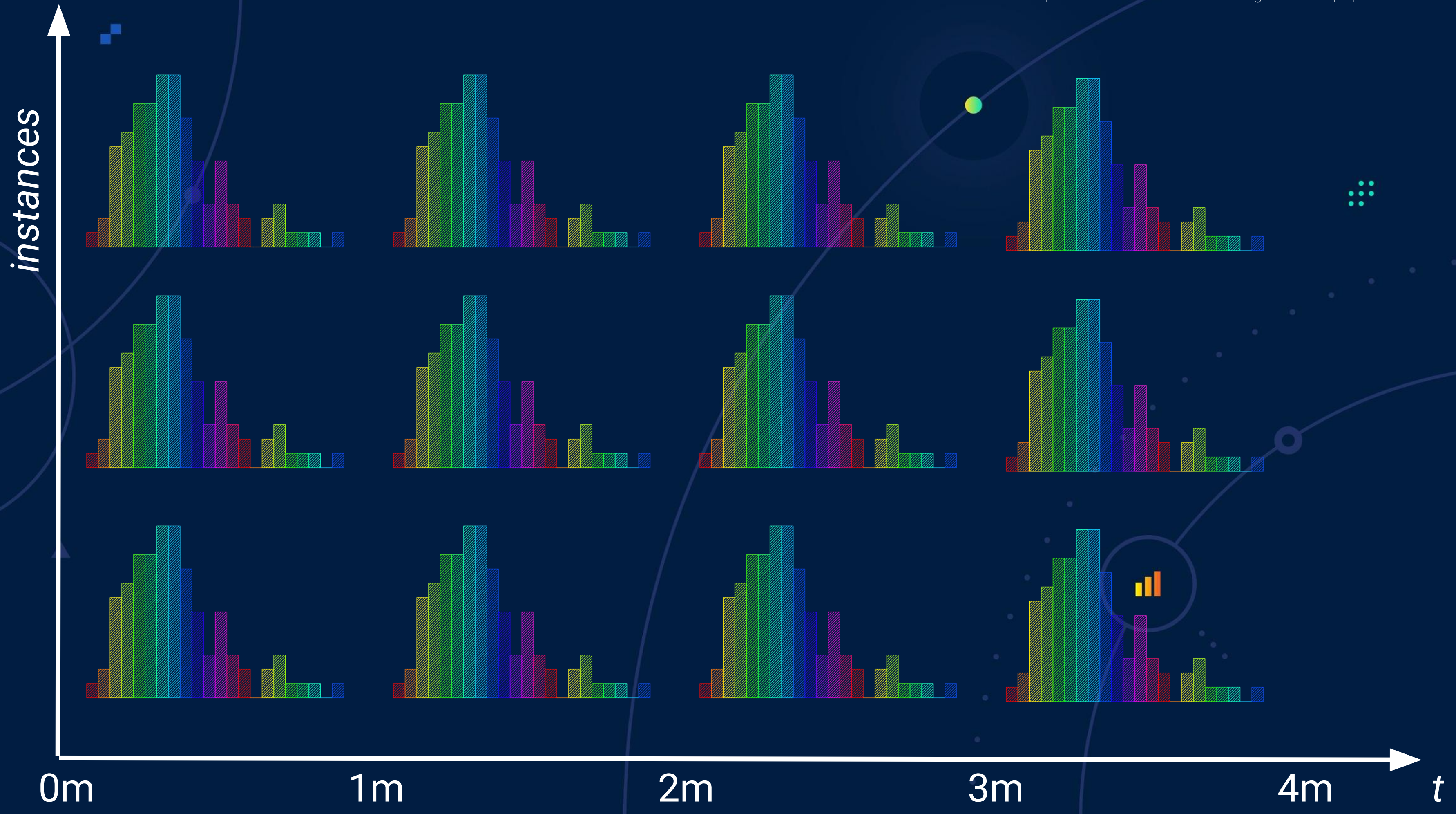
Histogram by DanielPenfield - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9401898



Histogram by DanielPenfield - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9401898



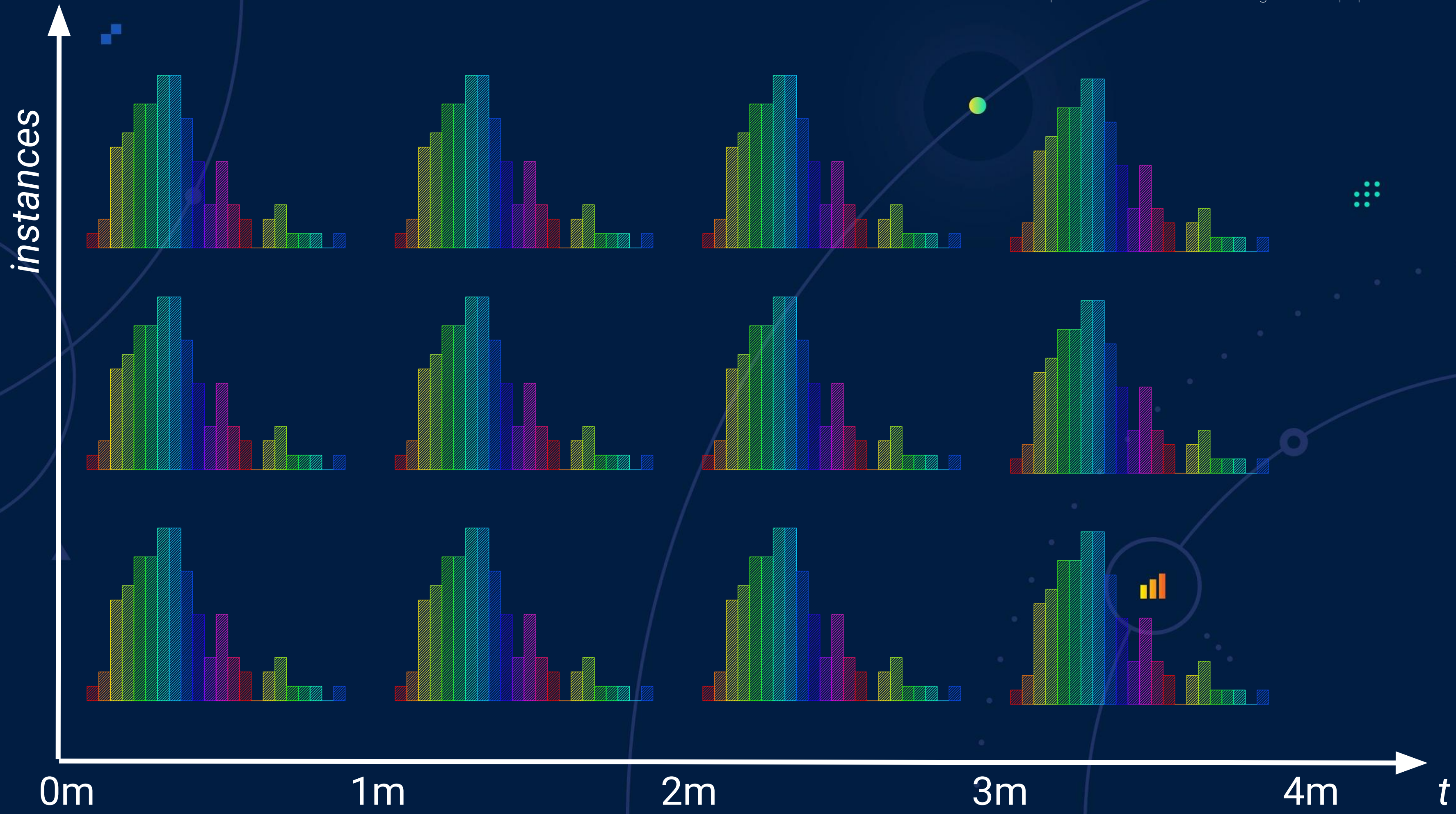
Histogram by DanielPenfield - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9401898

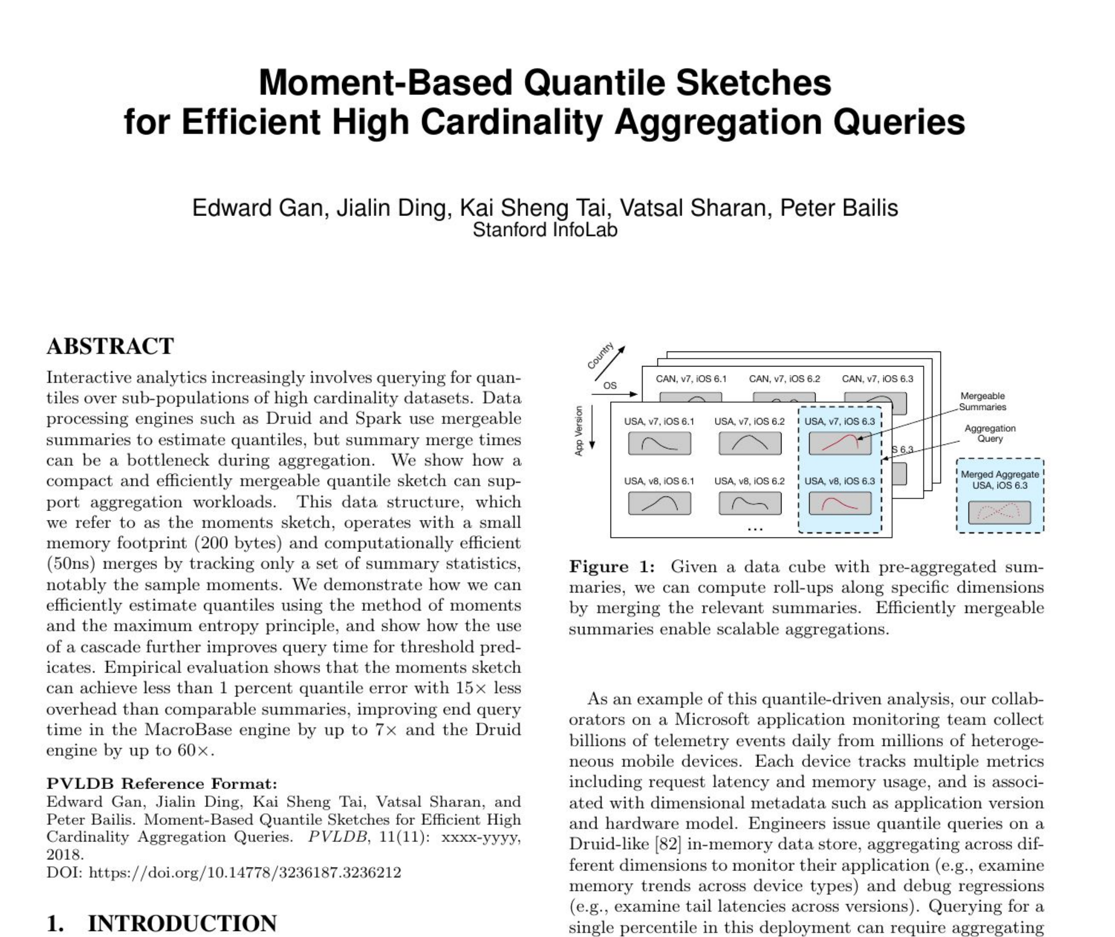


The Future

Option 4: Some kind of digest or sketch…

Histogram by DanielPenfield - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=9401898





Moment-Based Quantile Sketches for Efficient High Cardinality Aggregation Queries

Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, Peter Bailis
Stanford InfoLab

ABSTRACT

Interactive analytics increasingly involves querying for quantiles over sub-populations of high cardinality datasets. Data processing engines such as Druid and Spark use mergeable summaries to estimate quantiles, but summary merge times can be a bottleneck during aggregation. We show how a compact and efficiently mergeable quantile sketch can support aggregation workloads. This data structure, which we refer to as the moments sketch, operates with a small memory footprint (200 bytes) and computationally efficient (50ns) merges by tracking only a set of summary statistics, notably the sample moments. We demonstrate how we can efficiently estimate quantiles using the method of moments and the maximum entropy principle, and show how the use of a cascade further improves query time for threshold predicates. Empirical evaluation shows that the moments sketch can achieve less than 1 percent quantile error with 15× less overhead than comparable summaries, improving end query time in the MacroBase engine by up to 7× and the Druid engine by up to 60×.

PVLDB Reference Format:
Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. Moment-Based Quantile Sketches for Efficient High Cardinality Aggregation Queries. *PVLDB*, 11(11): xxxx-yyyy, 2018.
DOI: https://doi.org/10.14778/3236187.3236212

1. INTRODUCTION

Figure 1: Given a data cube with pre-aggregated summaries, we can compute roll-ups along specific dimensions by merging the relevant summaries. Efficiently mergeable summaries enable scalable aggregations.

As an example of this quantile-driven analysis, our collaborators on a Microsoft application monitoring team collect billions of telemetry events daily from millions of heterogeneous mobile devices. Each device tracks multiple metrics including request latency and memory usage, and is associated with dimensional metadata such as application version and hardware model. Engineers issue quantile queries on a Druid-like [82] in-memory data store, aggregating across different dimensions to monitor their application (e.g., examine memory trends across device types) and debug regressions (e.g., examine tail latencies across versions). Querying for a single percentile in this deployment can require aggregating

1.

2.

3.

4.

1.

2.

3.

4. Option 1: Do nothing.

1.
2.
3. Option 4: Digests/Sketches.
4. Option 1: Do nothing.

1.
2. Option 2: Make buckets a bit cheaper.
3. Option 4: Digests/Sketches.
4. Option 1: Do nothing.

1. Option 3: Master sparseness somehow.
2. Option 2: Make buckets a bit cheaper.
3. Option 4: Digests/Sketches.
4. Option 1: Do nothing.

https://github.com/beorn7/talks

beorn@grafana.com