一文搞懂 Prometheus 的直方图

histogram 的工作原理和分位数的计算方法

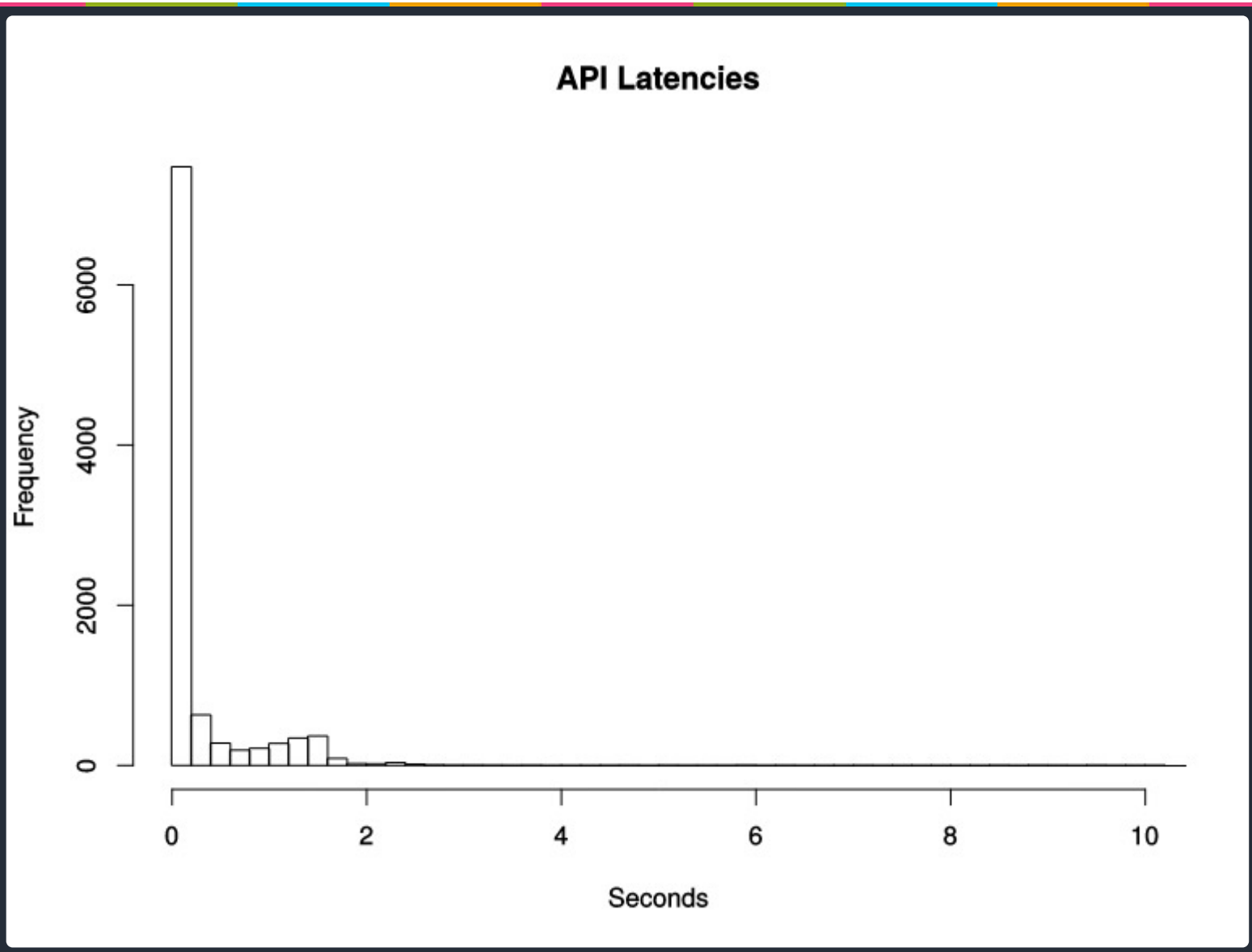
2019年08月06日 · 4 分钟 · 米开朗基杨 · 880 阅读 #prometheus #histogram

Prometheus 中提供了四种指标类型（参考： Prometheus 的指标类型），其中直方图（Histogram）和摘要（Summary）是最复杂和难以理解的，这篇文章就是为了帮助大家加深对这 `histogram` 类型指标的理解。

1. 什么是 Histogram?

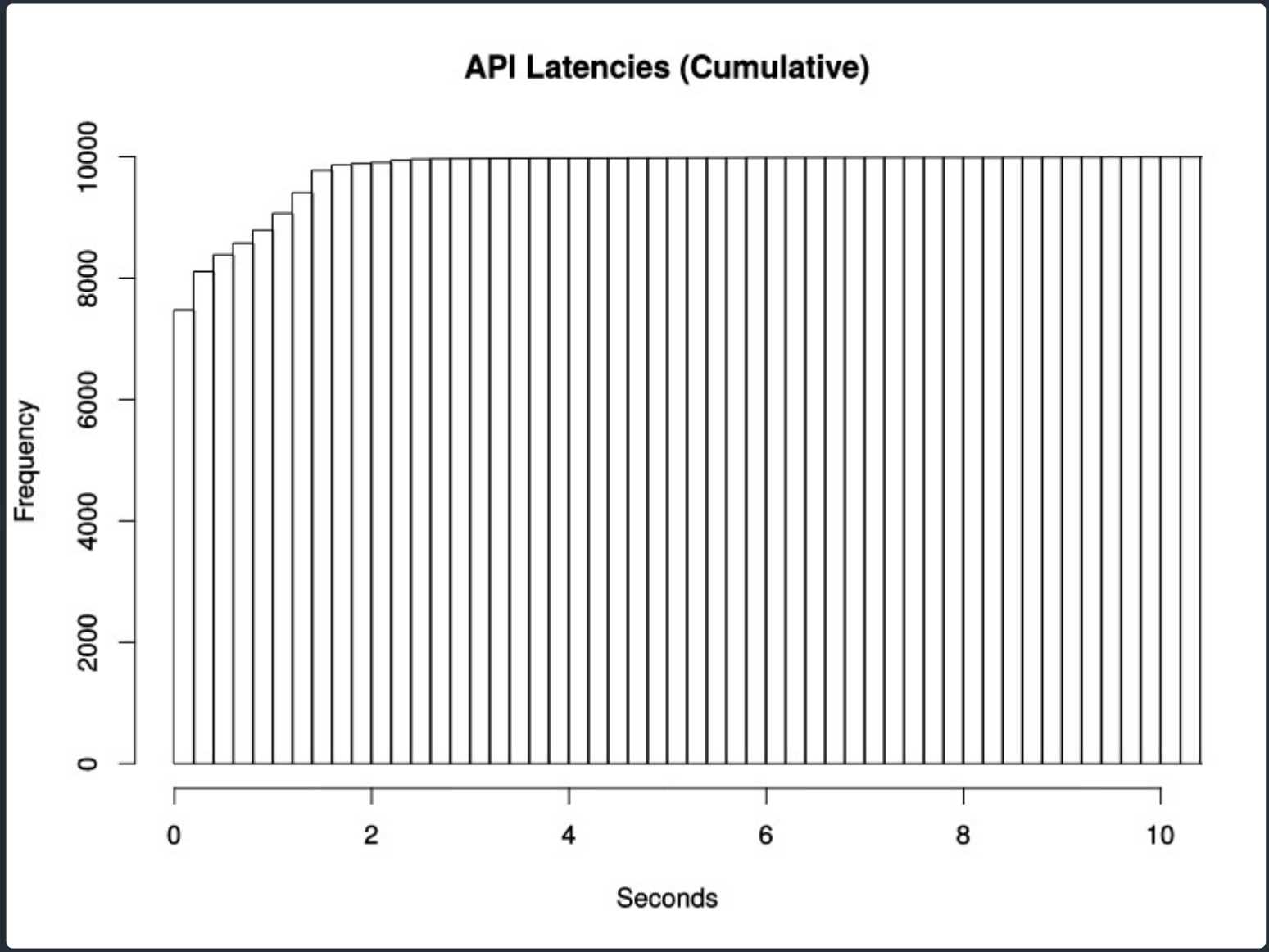
根据 上篇文档，Histogram 会在一段时间范围内对数据进行采样（通常是请求持续时间或响应大小等），并将其计入可配置的存储桶（bucket）中。但这句话还是不太好理解，下面通过具体的示例来说明。

假设我们想监控某个应用在一段时间内的响应时间，最后监控到的样本的响应时间范围为 0s~10s。现在我们将样本的值域划分为不同的区间，即不同的 `bucket`，每个 bucket 的宽度是 0.2s。那么第一个 bucket 表示响应时间小于等于 0.2s 的请求数量，第二个 bucket 表示响应时间大于 0.2s 小于等于 0.4s 的请求数量，以此类推。



Prometheus 的 histogram 是一种累积直方图，与上面的区间划分方式是有差别的，它的划分方式如下：还假设每个 bucket 的宽度是 0.2s，那么第一个 bucket 表示响应时间小于等于 0.2s 的请求数量，第二个 bucket 表

示响应时间小于等于 0.4s 的请求数量，以此类推。也就是说，每一个 bucket 的样本包含了之前所有 bucket 的样本，所以叫累积直方图。



2. 为什么是累积直方图?

上节内容告诉我们，Prometheus 中的 histogram 是累积的，这是很奇怪的，因为通常情况下非累积的直方图更容易理解。Prometheus 为什么要这么做呢?

想象一下，如果 histogram 类型的指标中加入了额外的标签，或者划分了更多的 bucket，那么样本数据的分析就会变得越来越复杂。如果 histogram 是累积的，在抓取指标时就可以根据需要丢弃某些 bucket，这样可以在降低 Prometheus 维护成本的同时，还可以粗略计算样本值的分位数。通过这种方法，用户不需要修改应用代码，便可以动态减少抓取到的样本数量。

假设某个 histogram 类型指标的样本数据如下:

```
# HELP example_latency_seconds Some help text
# TYPE example_latency_seconds histogram
example_latency_seconds_bucket{le="0.005"} 0.0
example_latency_seconds_bucket{le="0.01"} 0.0
example_latency_seconds_bucket{le="0.025"} 0.0
example_latency_seconds_bucket{le="0.05"} 1.0
example_latency_seconds_bucket{le="0.075"} 1.0
example_latency_seconds_bucket{le="0.1"} 1.0
example_latency_seconds_bucket{le="0.25"} 2.0
example_latency_seconds_bucket{le="0.5"} 3.0
example_latency_seconds_bucket{le="0.75"} 3.0
example_latency_seconds_bucket{le="1.0"} 4.0
example_latency_seconds_bucket{le="2.5"} 4.0
example_latency_seconds_bucket{le="5.0"} 5.0
example_latency_seconds_bucket{le="7.5"} 5.0
```

```
example_latency_seconds_bucket{le="10.0"} 5.0
example_latency_seconds_bucket{le="+Inf"} 5.0
example_latency_seconds_count 5.0
example_latency_seconds_sum 6.54
```

现在我们希望 Prometheus 在抓取指标时丢弃响应时间在 `100ms` 以下的 bucket，就可以通过下面的 relabel 配置来实现：

```
scrape_configs:
 - job_name: 'my_job'
   static_configs:
     - targets:
       - my_target:1234
   metric_relabel_configs:
   - source_labels: [ __name__, le ]
     regex: 'example_latency_seconds_bucket;(0\.0.*)'
     action: drop
```

其中，`example_latency_seconds_bucket` 用来匹配标签 `__name__` 的值，‘0.0.*’ 用来匹配标签 `le` 的值，即 `le` 的值为 `0.0x`。然后将匹配到的样本丢弃。

通过这种方法，你可以丢弃任意的 bucket，但不能丢弃 `le="+Inf"` 的 bucket，因为 `histogram_quantile` 函数需要使用这个标签。

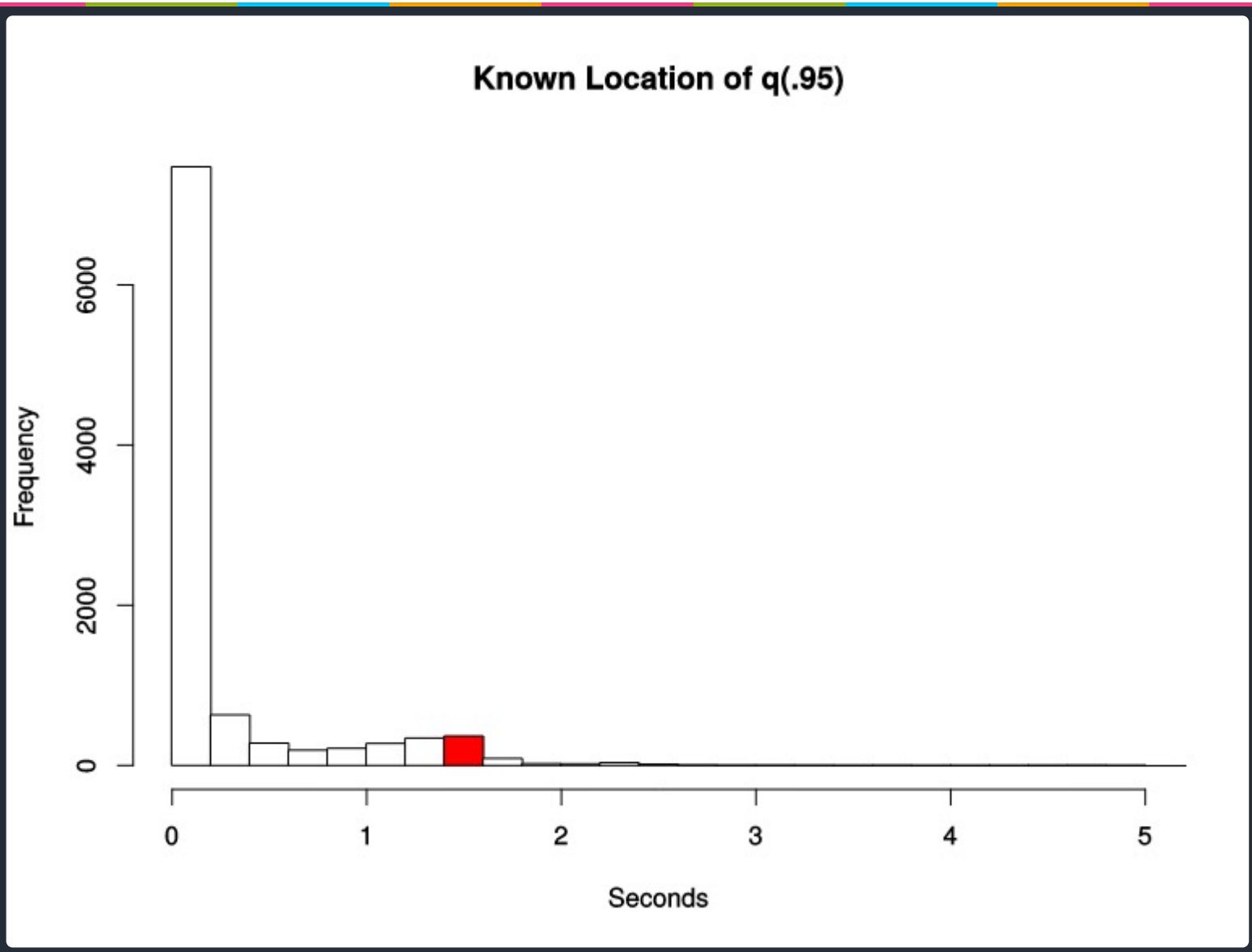
另外 histogram 还提供了 `_sum` 指标和 `_count` 指标，即使你丢弃了所有的 bucket，仍然可以通过这两个指标值来计算请求的平均响应时间。

通过累积直方图的方式，还可以很轻松地计算某个 bucket 的样本数占所有样本数的比例。例如，想知道响应时间小于等于 1s 的请求占所有请求的比例，可以通过以下公式来计算：

```
example_latency_seconds_bucket{le="1.0"} / ignoring (le) example_latency_seconds_bucket{le="+Inf"}
```

3. 分位数计算

Prometheus 通过 `histogram_quantile` 函数来计算分位数（quantile），而且是一个预估值，并不完全准确，因为这个函数是假定每个区间内的样本分布是线性分布来计算结果值的。预估的准确度取决于 bucket 区间划分的粒度，粒度越大，准确度越低。以下图为例：



假设有 `10000` 个样本，第 `9501` 个样本落入了第 8 个 bucket。第 8 个 bucket 总共有 `368` 个样本，其中第 `9501` 个样本在该 bucket 中属于第 `93` 个样本。

根据 Prometheus 源代码文件 promql/quantile.go 第 108 行的公式:

```
return bucketStart + (bucketEnd-bucketStart)*float64(rank/count)
```

我们可以计算 (quantile=0.95) 的样本值为：

$$(7 * 0.2) + 0.2 * \frac{93}{368} = 1.45054348$$

这个值已经很接近精确的分位数值了。关于 histogram_quantile 函数的详细使用方式，请参考： PromQL 内置函数。

4. 总结

本文主要介绍了 histogram 的工作原理以及分位数的计算方法，相信通过本文的抛砖引玉，大家应该对 Prometheus 的 histogram 有了更深一步的了解，下篇文章将会为大家呈现 Summary 的工作方式。

5. 参考资料

Prometheus and Histograms

-------他日江湖相逢 🌂 再当杯酒言欢-------