MENU

VividCortex

Why Percentiles Don’t Work the Way you Think

Posted by Baron Schwartz on Nov 18, 2016 2:16:00 PM

Customers ask us for p99 (99th percentile) of metrics pretty frequently.

We plan to add such a feature to VividCortex (more on that later). But a lot of the time, when customers make this request, they actually have something very specific, and different, in mind. They’re not asking for the 99th percentile of a metric, they’re asking for a *metric of 99th percentile.* This is very common in systems like Graphite, and it doesn’t achieve what people sometimes think it does. This blog post explains how percentiles might trick you, the degree of the mistake or problem (it depends), and what you can do if percentile metrics aren't right for you.

Image Credit

Away From Averages

Over the last few years a lot of people have started talking about the proble MENU h averages in monitoring. It's good that this topic is in wider discussion now, because for a long time averages were accepted without much deeper inspection.

Averages can be unhelpful when it comes to monitoring. If you're merely looking at averages, you're potentially missing the outliers, which might matter a lot more. There are two issues with averages in the presence of outliers:

1. Averages hide the outliers, so you can't see them.
2. Outliers skew averages, so in a system with outliers, the average doesn't represent typical behavior.

So when you average the metrics from a system with erratic behavior, you get the worst of both worlds: you see neither the typical behavior, nor the unusual behavior. Most systems have tons of outlying data points, by the way.

Looking at the extremes that lie in the "long tail" is important because it shows you how bad things can sometimes get, and you'll miss this if you rely on averages. As Amazon's Werner Vogels said in an AWS re:Invent keynote, the only thing an average tells you is that half of your customers are having a worse experience. (While this comment is totally correct in spirit, it isn't *exactly* right in practice: specifically, the median, or 50th percentile, is the metric that provides this property.)

Optimizely did a write-up in this blog post from a couple years ago. It illustrates beautifully why averages can backfire:

> "While the average might be easy to understand it's also extremely misleading. Why? Because looking at your average response time is like measuring the average temperature of a hospital. What you really care about is a patient's temperature, and in particular, the patients who need the most help."

Brendan Gregg also puts it well:

> "As a statistic, averages (including the arithmetic mean) have many practical uses. Properly understanding a distribution isn't one of them."

And Towards Percentiles

Percentiles (more broadly, quantiles) are often praised as a potential way to bypass this fundamental issue with averages. The idea of the 99th percentile is to take a population of data (say, a collection of measurements from a system) and sort them, then discard the worst 1% and look at the largest value that remains. The resulting value MENU vo important properties:

1. It’s the largest value that occurs 99% of the time. If it’s a web page load time, for example, it represents the worst experience that 99% of your visitors have.
2. It is robust in the face of truly extreme outliers, which come from all sorts of causes including measurement errors.

Of course, you don’t have to choose exactly 99%. Common alternate choices are 90th, 95th, and 99.9th (or even more nines) percentiles.

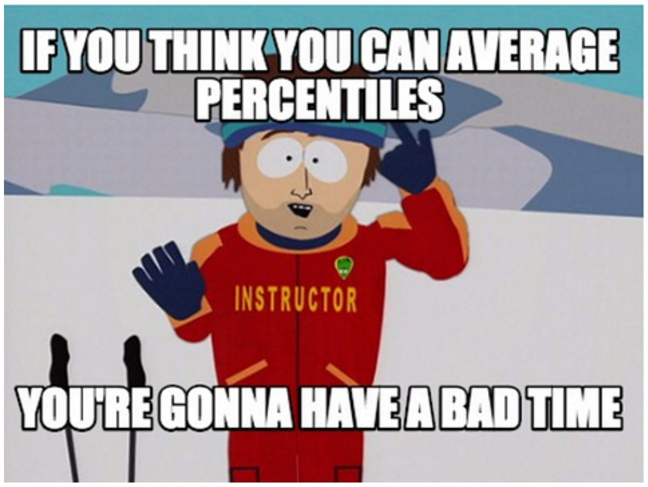
At this point, people assume: "averages are bad, and percentiles are great" — let’s calculate percentile metrics and put them into our time series databases, right? Not so fast.

How Time Series Databases Store and Transform Metrics

There’s a big problem with most time series data and percentiles. The issue is that time series databases are almost always storing *aggregate* metrics over time ranges, not the *full population* of events that were originally measured. Time series databases then *average* these metrics over time in a number of ways. Most importantly:

1. They average the data whenever you request it at a time resolution that differs from the stored resolution. If you want to render a chart of a metric over a day at 600px wide, each pixel will represent 144 seconds of data. This averaging is implicit and isn’t disclosed to the user. They ought to put a warning on that!
2. They average the data when they archive it for long term storage at a lower resolution, which almost all time series databases do.

And therein lies the issue. You’re still dealing with averages in some form. And averaging percentiles doesn’t work, because to compute a percentile you need the original population of events. The math is just broken. An average of a percentile is meaningless. (The consequences vary. I’ll return to that point later.)


IF YOU THINK YOU CAN AVERAGE PERCENTILES
INSTRUCTOR
YOU'RE GONNA HAVE A BAD TIME

MENU

A lot of monitoring software encourages the use of percentile metrics that are stored and resampled. StatsD, for example, lets you calculate metrics about a desired percentile, and will then generate metrics with names such as foo.upper_99 and emit those at intervals to be stored in Graphite. All well and good, if the time resolution you want to look at is never resampled, but we know that doesn't happen.

The confusion over how these calculations work is widespread. Reading through the related comments on this StatsD GitHub issue should illustrate that nicely. Some of these folks are saying things that just ain't so.

Perhaps the most succinct way to state the problem is this: *Percentiles are computed from a population of data, and have to be recalculated every time the population (time interval) changes. Time series databases with traditional metrics don't have the original population.*

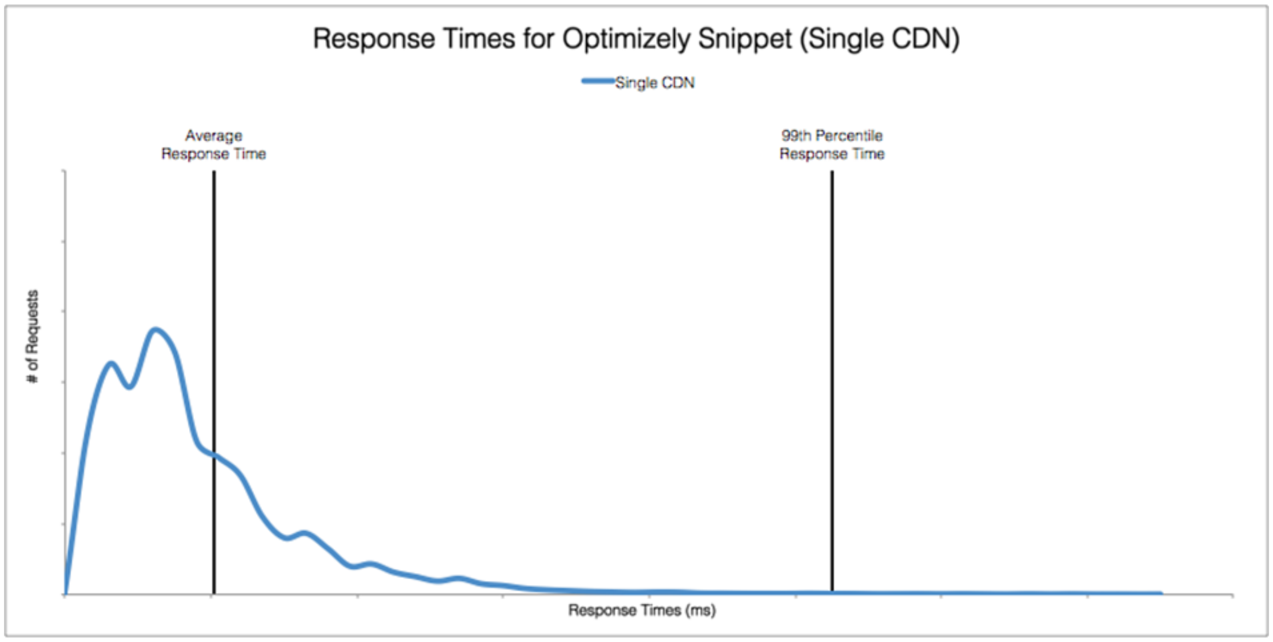
Alternative Ways To Compute Percentiles

If a percentile requires the population of original events—such as measurements of every web page load—we have a big problem. A Big Data problem, to be exact. Percentiles are notoriously expensive to compute because of this.

There are lots of ways to compute *approximate* percentiles that are almost as good as keeping the entire population and querying and sorting it. You can find tons of academic research on a variety of techniques, including:

- Histograms, which partition the population into ranges or bins, and then count how many fall into various ranges.
- Approximate streaming data structures and algorithms (sketches).
- Databases that sample from populations to give fast approximate answers.
- Solutions that are bounded in time, bounded in space, or both.

The gist of most of these solutions is to approximate the *distribution* of the MENU tion in some way. From the distribution, you can compute at least the approximate percentiles, as well as other interesting things. From the Optimizely blog post, again, there's a nice example of a distribution of response times and the average and 99th percentile:



Source: Catchpoint.com, data from Oct. 15, 2013 to Nov. 25, 2013 for 30KB Optimizely snippet.

There are tons of ways to compute and store approximate distributions, but histograms are popular because of their relative simplicity. Some monitoring solutions actually support histograms. Circonus is one, for example. Circonus's CEO Theo Schlossnagle often writes about the benefits of histograms.

Ultimately, having the distribution of the original population isn't just useful for computing a percentile, it's very revealing in ways that the percentile isn't. After all, a percentile is a single number that tries to represent a lot of information. I wouldn't go as far as Theo did when he tweeted that "99th percentile is as bad as an average," because I agree with percentile fans that it's more representative of some important characteristics of the underlying population than an average is. But it's not as representative as histograms, which are much more granular. The chart above from Optimizely contains way more information than any single number could ever show.

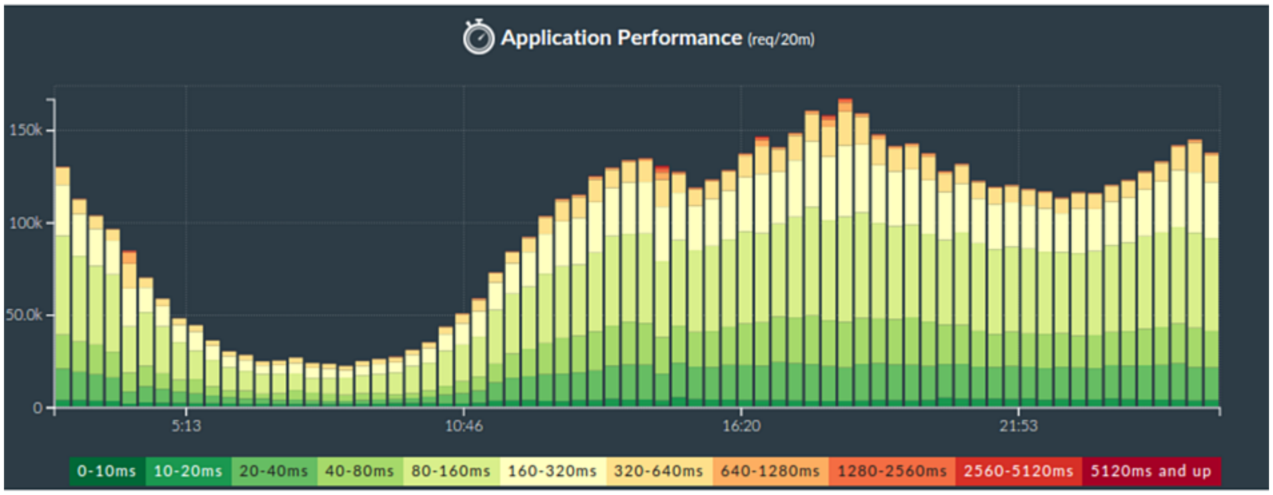
Percentiles Done Better in Time Series Databases

A better way to compute percentiles with a time series database is to collect banded metrics. I mention the assumption because there are lots of time series databases that are just ordered, timestamped collections of named values, without the capability of storing histograms.

Banded metrics provide a way to get the same effect as a series of histogra MENU r time. What you'd do is select limits that divide the space of values up into ranges or bands, and then compute and store metrics about each band over time. The metric will be just as it is in histograms: the count of observations that fall into the range.

Choosing the ranges well is a hard problem, in general. Common solutions include logarithmic ranges and ranges that provide a given number of significant digits but may be faster to calculate at the cost of not growing uniformly. Even divisions are rarely a good choice. For more on these topics, please read Brendan Gregg's excellent writeup.

The fundamental tension is between the amount of data retained and the fineness of the resolution. However, even coarse banding can be effective for showing more than simple averages. For example, Phusion Passenger Union Station shows banded metrics of request latencies using 11 bands. (I don't think the visualization is the most effective; the y-axis's meaning is confusing and it's essentially a 3d chart mapped into 2d in a nonlinear way. Nevertheless, it still shows more detail than an average would reveal.)



How would you do this with popular open source time series tools? You'd have to define ranges and create stacked charts as shown.

To compute a percentile from this would be much more difficult. You'd have to range over the bands in reverse order, from biggest to smallest, summing up as you go. When you reach a sum that's no more than 1% of the total, that band contains the 99th percentile. There are lots of nuances in this—strict inequalities, how to handle edge cases, what value to use for the percentile (upper or lower bin limit? in the middle? weighted?).

And the math can be confusing. You might think, for example, that you need at least 100 bands to compute the 99th percentile, but it depends. If you have 2 bands and the uppermost band's value contains 1% of the values, you've got your 99th percentile. (If that sounds counterintuitive, take a moment to ponder quantiles; I think a deep understanding of quantiles is worthwhile.)

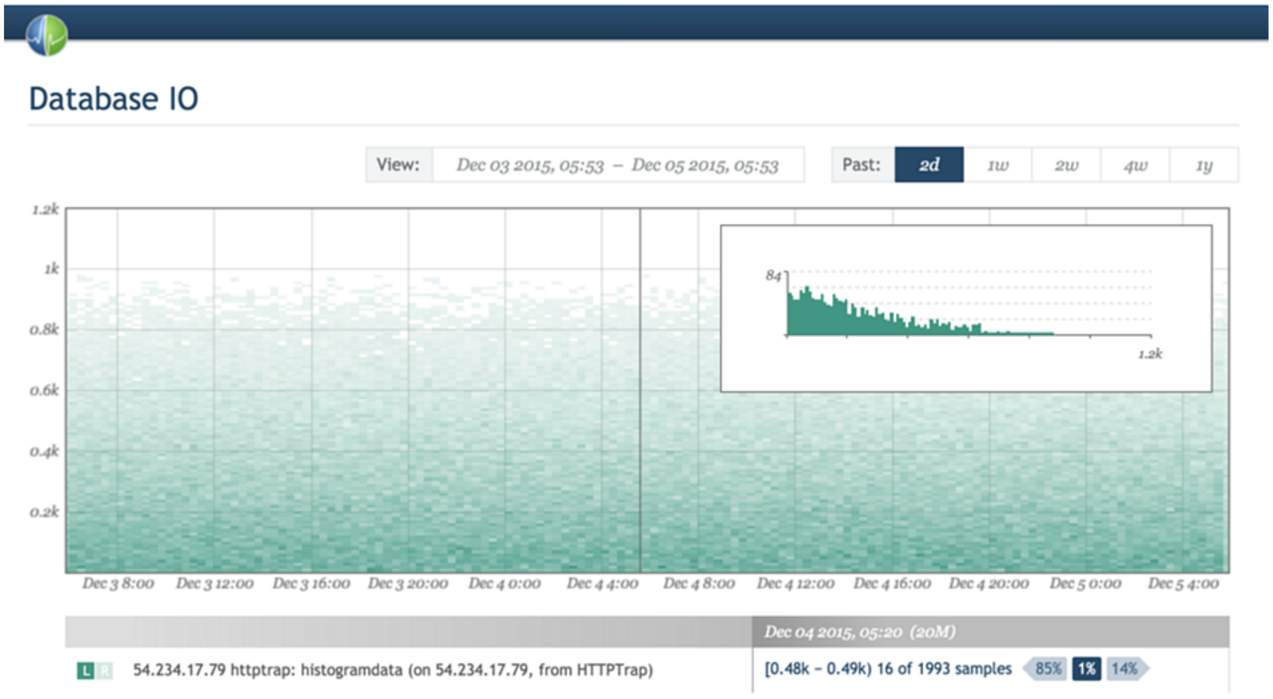
So this is complicated. It's possible in the abstract, but it also largely depend MENU whether a database's query language supports the calculations you'd need to get an approximate percentile. If you can confirm systems in which this is definitely possible, please comment and let me know.

The nice thing about banded metrics in a system like Graphite, which treats all of its metrics naively in terms of assuming they can be averaged and resampled at will, is that banded metrics are robust to this type of transformation. You'll get correct answers because the calculations are commutative over all time ranges.

Beyond Percentiles: Heatmaps

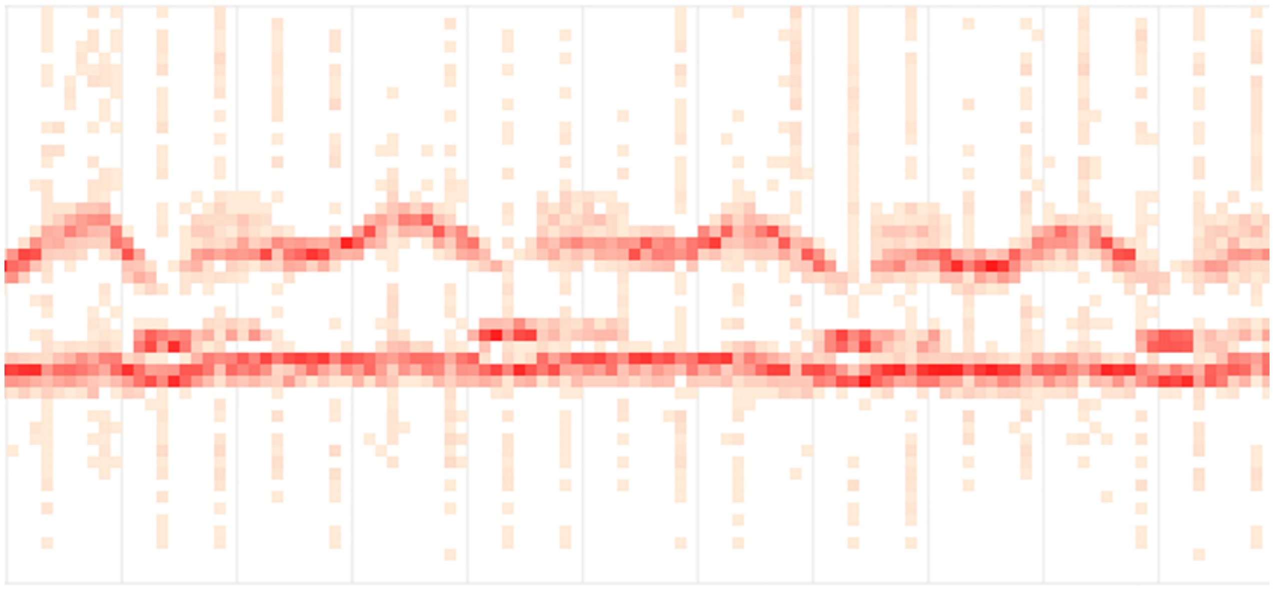
A percentile is still a single number, just like an average. An average shows the center of gravity of a population, if you will; a percentile shows a high-water mark for a given portion of the population. Think of percentiles as wave marks on a beach. But although this reveals the boundaries of the population and not just its central tendency as an average does, it's still not as revealing or descriptive as a distribution, which shows the shape of the entire population.

Enter heatmaps, which are essentially 3d charts where histograms are turned sideways and stacked together, collected over time, and visualized with the darkness of a color. Again, Circonus provides an excellent example of heatmap visualizations.

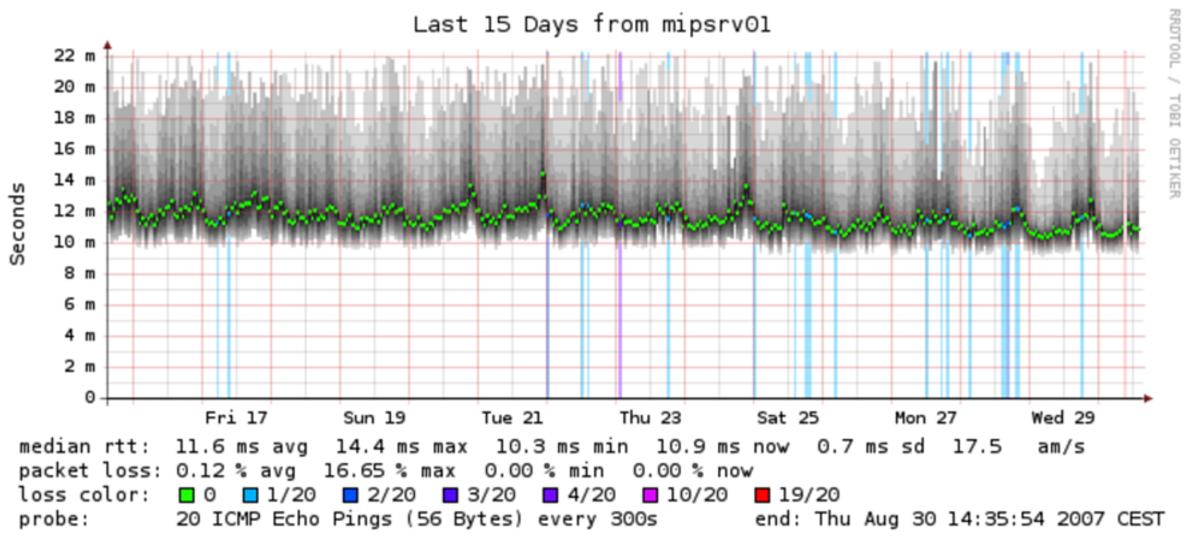


On the other hand, as far as I know, Graphite does not have the ability to produce heatmaps with banded metrics. If I'm wrong and it can be done with a clever trick, please let me know.

Heatmaps are great for visualizing the shape and density of latencies, in par MENU. Another example of heatmap latency is Fastly's streaming dashboard.



Even some old-fashioned tools that you might think of as primitive can produce heatmaps. Smokeping, for example, uses shading to show the range of values. The bright green is the average:



```
median rtt:   11.6 ms avg   14.4 ms max   10.3 ms min   10.9 ms now   0.7 ms sd   17.5   am/s
packet loss:  0.12 % avg   16.65 % max   0.00 % min   0.00 % now
loss color:   🟩 0   🟦 1/20   🟦 2/20   🟪 3/20   🟪 4/20   🟪 10/20   🟥 19/20
probe:        20 ICMP Echo Pings (56 Bytes) every 300s          end: Thu Aug 30 14:35:54 2007 CEST
```

How Bad Is It To Store Metrics Of Percentiles?

OK, so after all this complexity and nuance, perhaps the good old StatsD upper_99 metrics of percentiles aren't sounding so bad anymore. After all, they're pretty simple and efficient, and it's a turnkey solution. How bad are they, really?

It depends. For lots of purposes they're absolutely fine. I mean, you still have the limitations that percentiles aren't very descriptive by themselves, and all that. But if you're okay with that, then the biggest issue remaining is how they get mangled by resampling, which basically means you're looking at wrong data.

But all measurements are wrong anyway, and besides, lots of wrong things MENU eful regardless. For example, I'd say that half of the metrics people use in monitoring systems are already deliberately transformed in ways that mangle them. Load average, for example. It's very useful, but when you realize how the sausage is made, you might be a little shocked at first. Similarly, lots and lots of widely available systems report partially digested metrics about their performance. A bunch of Cassandra's metrics are outputs from Coda Hale's Metrics library, and are time-decayed averages (exponentially weighted moving averages), which some people have a huge aversion to.

But back to metrics of percentiles. If you store a p99 metric and then zoom out and view an averaged version over a long time range, although it won't be "right," and may be quite different from the actual 99th percentile, the ways in which it is wrong won't necessarily render it unusable for the desired purpose, i.e. understanding the worst experience most of your users are having with your application. Regardless of their exact values and how wrong they are, percentile metrics tend to a) show outlying behavior and b) get bigger when outlying behavior gets badder. Super useful.

So it depends. If you know how percentiles work and that averaging a percentile is wrong, and you're okay with it, it might still be useful to store metrics of percentiles. But you are introducing a sort of moral hazard: you might deeply confuse people (perhaps your colleagues) who don't understand what you've done. Just look at the comments on that StatsD issue again; the confusion is palpable.

If you'll permit me to make a bad analogy, I'll sometimes eat and drink things in my fridge that I'd never give to someone else. (Just ask my wife.) If you give people a bottle labeled "alcohol" and it contains methanol, some of them will drink it and go blind. Others will ask "what kind of alcohol is in this bottle?" You need to bear that responsibility.

What Does VividCortex Do?

At the moment, our time series database doesn't support histograms, and we don't compute and store metrics of percentiles (although you can easily send us custom metrics if you want).

In the future, we plan to store banded metrics at high resolution, i.e. lots of bands. We can do this because most bands will probably be empty, and our time series database handles sparse data efficiently. This will essentially give us histograms once per second (all of our time series data is 1-second granularity). We downsample our data to 1-minute granularity after a configurable retention period, which is 3 days by default. Banded metrics will downsample into 1-minute-granularity histograms without any mathematical curiosities.

And finally, from these banded metrics, we'll be able to compute any desired MENU entile, indicate the estimated error of that value, show heat maps, and show distribution shapes.

This won't be a quick project and will require lots of engineering in many systems, but the foundation is there and we designed the system to eventually support this. No promises on when we'll get it, but I thought it'd be useful to know where our long-term thinking is.

Conclusions

This was a longer post than I thought it'd be, and I covered a lot of ground.

- If you want to compute percentiles at intervals and then store the results in a time series database — as some extant databases currently do — you might not be getting what you think you are.
- Real percentiles require massive amounts of data processing.
- Approximate percentiles can be computed from histograms, banded metrics, and other useful techniques.
- These datasets also enable distributions and heatmaps, which are much more information-rich than percentiles.
- If this is out of your reach at the moment, go ahead and use metrics of percentiles, but know the consequences.
- Metrics of percentiles tend to a) show outlying behavior and b) get bigger when outlying behavior gets badder, which is useful for lots of purposes.

Hopefully this has been helpful. Also, if you'd like to see some of VividCortex's approaches and solutions to these problems, it's easy to **start your free trial** of VividCortex. Don't hesitate to get started today.

P.S.

- Someone commented on Twitter to the effect of, "oh interesting, I'm doing it wrong. I'll switch to calculating the percent of requests that are over/under a desired latency and store that metric instead." This doesn't work either. Averages of fractions (a percent is a fraction) don't work. Instead, store a metric of the *number* of requests that didn't meet your desired latency. That'll work ok.
- I vaguely remembered but didn't find Theo's excellent post on a related topic. Here it is: http://www.circonus.com/problem-math/

Tweet Share Like 237 Share

Topics: Math and Statistics

MENU

Troubleshoot and Diagnose Outages

Sign up for your Free Trial!
Immediately uncover potential issues that coincided with an outage. Quickly pinpoint where to investigate to find the root cause so that your team can make changes, recover faster, and avoid future problems.

First Name	Last Name
Email	
Phone Number	

☐ I agree to VividCortex's Terms of Service and Privacy Policy.

GET MY FREE TRIAL

Recent Posts

- The Marvel of Observability
- EnterpriseDB Joint Webinar: Best Practices for Monitoring Postgres
- What Changed? Find Out With Explorer Time Comparison
- Create Post-Mortem Reports from Notebook Templates, Export to PDF
- Hide VividCortex's Queries From The Profiler and Explorer
- Postgres EXPLAIN Explained
- Configure Alerts on Deadlock Events
- Learn How To Connect Golang to Databases with Database/SQL
- Manage Database Incidents with VividCortex and Opsgenie

MENU

VividCortex Achieves SOC 2 Type II Certification

Posts by Topic

- Monitoring (149)
- Product Features (91)
- MySQL (54)
- Culture (48)
- News, Press and Media (46)

see all