iVoyeur
Approaching Normal

DAVE JOSEPHSEN

Dave Josephsen is the author of Building a *Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project. dave-usenix@skeptech.org

In the past, I have been known to indulge in hyperbole. I freely admit this, but to say that our Christmas tree erupted into flames wouldn't come close to an adequate description. It exploded into a wholly new form of matter that transcended flames. I tell you, one moment it was a chopped up, dead Christmas tree in a fire-pit, and the next moment it was a churning column of violent, malevolent living fire. It was as if there were something inside that tree. As if the tree had been imbued with some small piece of the soul of Christmas, or maybe anti-Christmas, and we had set it alight. The tree had arms, and a face, and it writhed and screamed and reached out for us, beckoning for us to come closer. We took a few steps back.

"Is that normal?" my wife asked, not taking her eyes off the 15-foot columnar fire-being.

Normal; I paused to consider. To *know* whether a thing is normal, we'd have to define normal, by quantitative measurement and comparison. To honestly *know* the truth of the "normal" interaction between Christmas trees and matches, we'd have to burn every Christmas tree on earth to establish a mean, and compute our Christmas tree's standard deviation from that mean. Then we could objectively say whether or not this was "normal." This is probably an impossible task (but I'm willing to try if you're able to fund me).

"I don't know." I replied, taking note of the location of the garden hose.

Of course, it isn't true that we'd need to burn every Christmas tree on earth. There are many problems that require us to quantify the "normalness" of a given property in a population. And it's not unusual for the population to be large enough to make infeasible the measuring of every member. All we really need is a statistically relevant sample from which we can estimate the normal interaction between Christmas trees and matches.

By "statistically relevant," I'm referring not only to our ability to accurately approximate the normal interaction between Christmas trees and matches, but also to our ability to compute the accuracy of our estimates. Statistics was invented to answer just this sort of (ahem) burning conundrum. Really, this is a data sampling problem of the sort that is important in many systems monitoring and analytics contexts, and will only become more important as the data grows. Further, sampling can help us scale monitoring systems, not only by reducing the amount of measuring we have to do, but also by reducing the metric data we need to transmit.

The Simple Random Sample

Most of what humanity knows about the science of data sampling comes from statistics problems involving subjective human opinions on this or that political issue. Even when the problems involve "harder" subjects, such as crime statistics, our humanity can intrude in ways that make it notoriously difficult to acquire data samples that are statistically relevant. For this reason there are myriad data sampling methodologies that we can ignore entirely in a monitoring context. For our purposes, we will usually be looking for a "simple random sample," or "SRS" [1].

Taking humans out of the sample-collecting picture (mostly) helps us to avoid the common pitfalls that have a negative impact on the accuracy of conclusions statistically derived from sampled data. Selection bias, overcoverage, undercoverage, processing errors, and measurement errors are all out the window. This makes data sampling a really effective technique for the kinds of problems we're usually concerned with in a monitoring context.

Gleaning a simple random sample is, well, simple. We just need to choose a subset of measurements from a larger body of data in a way that every measurement has an equal probability of being chosen as a sample. Imagine, for example, that you've written a program that calls a function foo(). You want to monitor the length of time foo() takes to return, but your program is distributed across several hundred compute nodes, so foo() gets called several million times per second. You could obtain an SRS for foo() return times by generating a pseudo-random number every time foo() is called, and comparing it to a threshold. If the current random number is less than the threshold, then you take a sample return time. If that's too expensive for your purposes, the random requirement can usually be adequately satisfied in practice by simply decrementing a counter and sampling when it hits zero (especially in large populations like foo()).

Sample Size

If we generated a random number between 0 and 1, and used .0025 (a quarter of one percent) for our threshold, we should collect around 2,500 samples for every million calls to foo(). The Central Limit Theorem [2] allows us to make a few assumptions about our sample. For instance, we can assume that our sample mean pretty much equates to our population mean. By "pretty much" I mean our sample approximates the population within some margin of error, which we can compute [3]. In my example our sample should approximate the population within about a 5% margin of error.

As you might expect, if we reduce the sample size, we increase the error percentage. This relationship directly affects the accuracy of our estimates, and means we can increase accuracy by either sampling more often, or sampling for a longer period of time. It also means that it is important to deliberately choose your sample size [4].

Here's something you might not expect: as long as the population is an order of magnitude larger than the sample, the accuracy of our predictions does not vary with the size of the population. In other words, it wouldn't matter if foo() was being called a million times per second, or ten thousand. My 2,500 samples would give me the same accuracy in either case.

That's weird but useful; it means we don't need to think about data samples as a percentage of the total population for many of the problems we're interested in, which certainly helps us scale. It also introduces the irony that smaller populations are more difficult to sample accurately.

Sometimes, Sampling Ruins Everything

sFlow [5] is a great example of a controversial use of data sampling. Sometimes we're interested in knowing the number of occurrences of X in a population, like the number of packets that make up the total population of packets that traversed a given switch port, or the number of bytes sent that were BitTorrent protocol in the population of all bytes.

These numbers are expensive (in a computational sense) to gather and process. Traditional approaches, such as hardware packet taps, span ports, and NetFlow-enabled switches, burn every Christmas tree by either measuring each packet directly or copying each packet to an external entity. The cost of this brute-force endeavor is actualized as more expensive (in dollars and cents) network gear, or slower network gear.

sFlow, by comparison, gleans a simple random sample by decrementing a counter per interface, and sampling the current packet when the counter hits zero. By modeling the sample as a binomial distribution [6] sFlow can, at near zero cost, answer questions like the BitTorrent-related ones above with sufficient accuracy for the purpose of customer billing in the real world. This is clever engineering, and the sFlow creators have obviously [7] put careful thought into its design and implementation. The accuracy of its estimates are guaranteed by math.

What sFlow cannot guarantee, however, is that all classes of traffic actually make it into the sample set. It's entirely feasible that small, one-time bursts of traffic (the packets making up a port scan, for example) might never be sampled, and therefore never be represented in sFlow's output (and this property, by the way, does vary with the size of the population). So while flow statistics of the sort that are interesting to network operations folks are accurately represented, the kind of statistically aberrant thing that might interest the security folks is maybe not.

The Buddha said that it's not what we carry with us but what we let go of that defines us, which is an apropos sentiment in this case. I'm sure it was not a design goal for sFlow to capture these aberrant, one-off micro-flows, but their exclusion renders sFlow unusable for a huge chunk of the reason many of us run packet taps, span ports, and NetFlow, which is to say, intrusion detection and forensics, and, therefore, since we're doomed to incur the cost of those other solutions anyway, belies our use of sFlow entirely. That's kind of sad because I personally like clever engineering, statistics, and things that are cheap (in any sense), but I also think it's possible that data sampling and traffic inspection might not be compatible undertakings.

It's easy, especially for statistics geeks, to become overly fascinated with the center mean line of that perfectly symmetrical bell curve: the normal distribution. When we plot a sample that approximates the normal distribution for some attribute, we know that we've methodologically nailed it. But the prudent monitoring engineer should, in many endeavors, be concerned with the statistically irrelevant, with whether the flaming Christmas tree is, in fact, abnormal instead of normal. Even if the cost is burning them all, it is a cost that should be weighed against the loss of statistically irrelevant but really interesting weirdo observations.

Take it easy.

References

[1] The simple random sample: http://www.ma.utexas.edu/users/mks/statmistakes/SRS.html.

[2] The Central Limit Theorem: http://en.wikipedia.org/wiki/Central_limit_theorem.

[3] Sample means: http://www.stat.yale.edu/Courses/1997-98/101/sampmn.htm.

[4] Choosing a sample size: http://www.itl.nist.gov/div898/handbook/ppc/section3/ppc333.htm.

[5] sFlow: http://www.inmon.com/technology/index.php.

[6] sFlow packet sampling basics: http://www.sflow.org/packetSamplingBasics/index.htm.

[7] sFlow sampling theory: http://www.sflow.org/about/sampling_theory.php.