iVoyeur

Counters

DAVE JOSEPHSEN

Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

My friend Chris counts people. Well, sometimes he counts people, and although I can tell you where, I couldn't tell you why. I only found out by accident, and the act of discovery felt enough like an unwelcome intrusion that asking *why* seemed out of the question.

It happened when a group of us went over to his apartment at lunch. He lived across the street from the building in which we all worked at the time, and it wasn't uncommon for us to grab a sandwich or whatever, and head over to his apartment to play some "DOA3" or "God Hand." On this particular day, Chris had left on his kitchen table a small metallic four-digit "hand tally," like the ones you see in the hands of parking attendants and people who take your ticket as you come through the door.

My friend Kelly, who, like myself, has a penchant for sometimes accidentally asking embarrassing questions, picked it up and, glancing at the number (37 IIRC), asked what it was. Chris replied, in so many words, that the number represented people that he talked to at a party the night before, and didn't appear eager to discuss it much further.

We said nothing more about it. But I often wish that Kelly had had the lack of decorum to pursue it, or that I had, because I don't know about you, but thousands of questions occur to me. Important, nagging questions that, having gone unanswered, keep this story in my memory year after year, even though nearly a decade has passed. When I die, I will die with these questions somewhere in what remains of my dementia-riddled mind. Was this a recurring polling interval, or was it a non-periodic metric? Was he storing it as an incrementing counter that rolled over at 9999+1, or was he manually rolling it over daily and storing it as a gauge? Had he plotted the distribution? Was it uniform? What was the mean and sigma? Had he done any predictive analysis?

I was tempted to begin this article with something like, "Counters are a foundational concept in systems monitoring," although it would have been beyond condescending of me. You don't need me to tell you that counting things is a foundational concept. Counters are about as simple a concept as exists. Yet, in the context of this story about Chris, it strikes me that counter metrics are also sort of subtle and sometimes misunderstood.

Indeed the questions I have about Chris's quantification fetish that I cannot answer have mostly to do with the implementation details of the counter data. I can, for example, imagine myriad reasons why he might want to count the people he talks to—that he has a personal goal to talk to 10k unique people, or is trying to improve his interpersonal skills by repetition. I've often fallen down this or that "life hacking" rabbit hole and can easily relate. But the details surrounding how his counter metric is stored imply real and interesting systems constraints.

For example, Chris, like me, cut his teeth on MRTG and RRDtool, so for him to store his metric as type "counter" in a round-robin database would be natural. RRDtool's concept of a counter grew out of SNMP counters. Network devices, such as switches and routers, store metrics, such as byte-counters, on each interface using a 32-bit number. RRDtool's counter type is intended to store the current value polled from a router byte-counter directly and automatically compute and display the derivative, or rate of change, of that value. When you say "store it as a counter" to Ops people, this is the notion you have conjured in their minds—that of an ever-increasing 32- or 64-bit value, which, if plotted directly, would always look like a diagonal line increasing from left to right.

In the mind of a programmer, a counter is also usually a 32-bit value, although a good programmer would probably disown that, muttering something about it being architecture dependent, and something else about uint32. The programmer's counter usually has a short name (like "c") and the programmer usually has handy incrementors and decrementors (like "++" and "--") built into his or her favored language that enable simple and terse manipulation of the counter value.

If my friend Chris tired of having a manual clicky thing in his pocket that he had to interact with, and then later, manually copy the value from, we can imagine that he might replace his hand-tally with something like an Arduino board that could uniquely identify the voices of the people he talked to. If he did this, internally, he would probably use a 32-bit integer with a small name (like "p") in his code to track his "people" metric. This would be good, but he would still want to store the metric externally, so he could see graphs without manually copying values from the device. The Arduino could use WiFi (or whatever) to broadcast his metric every five or ten minutes, which would enable him to see not only how many people he spoke to but also when he did a lot of talking over the course of the evening.

He could store the value externally in Graphite [1], a metrics collection system with a super-simple API. Some monitoring systems, such as Graphite, only store data as a value that may increase or decrease (a "gauge," in RRDtool parlance). If you want a useful graph of a counter stored as a gauge in Graphite, you wrap your counter in the "derive()" function when you graph it. So if Chris wants to store his metric in Graphite, he'll need to write his code such that the total value of p is sent to Graphite every so often, like RRDtool with the SNMP byte-counters.

Incrementor operators such as "++" are ubiquitous, easy to understand, and make a lot of sense in contexts like counting people. If, in Chris's code, instead of sending the value of p every time, it would be nice if he could create some sort of object, or type, which he could simply increment each time he had a new conversation with a unique person. Graphite's socket API is nice, but because it doesn't have a counter type at all, it doesn't provide a simple means for Chris to, in his code, say "p++". Chris would either have to track the current total value himself and then write a wrapper for himself to increment and push it, or he'd have to query Graphite for the current total value, increment it, and then push it back.

Either way, Chris has scalability details to consider. If he tried to track the total value himself, he would limit concurrency. If, for example, he eventually attempted to replace his Arduino device with a series of room-based devices, each of which reported more generic stats on all conversations in their respective room, each device would have a different total value of p for Chris, and would therefore step on each other when they tried to report up to Graphite. If Chris tried to query the current value for p for every measurement, he'd have a race condition, limit the polling interval, and introduce a dependency between metrics collection and storage, which is an unwise design choice.

Instead of communicating values, if Chris's code could send an Incrementor signal to the external storage system would be ideal. That way, he wouldn't need to store or acquire the current value of p to increment it, and his code could be distributed and concurrent. Scalability concerns would be pushed up from the metrics collection tier to the metrics storage tier, and the number of bits on the wire would be smaller and more predictable. This, in a nutshell, is why Statsd [2] was created. Statsd was designed to be a middle layer between Graphite and your metrics sources. It can listen for Graphite protocol values as well as special purpose instructions, such as incrementor signals. Statsd can also "roll up" multiple broadcasts from metrics collectors into single updates, taking some of the strain off Graphite itself.

In case I haven't made the subject of counters convoluted enough, to run Statsd on every server, or centrally, or both is common practice. We can imagine, for example, Chris's multi-room conversation-counter system, tracking per-room metrics, in which case each room would have its own value of p that would be tracked as a separate metric. If we wanted a total value, we could sum() them at visualization time in Graphite. Alternatively, we could run Statsd at a central location, and point every room at it. We could still track per-room metrics this way, as well as allowing every server to increment a global total value counter. Finally, we could run Statsd on every server, which could give us a high localized polling interval, and then point each server Statsd instance at a centralized Statsd, to roll everything up and maintain a global counter.

In my opinion, the programmatic notion of a counter as "a value we use to count things," is more intuitive than RRDtool's "ever-increasing, sometimes rolling-over, value we assume you're going to want to plot the derivative of at some point." More importantly, the two definitions are not compatible with one another in a data-storage sense. We cannot store the former in a box designed for the latter.

For example, Chris might eventually run into some situations in which he wants to decrement p, especially if every conversation is automatically being counted for him. "Conversations" beginning "Where is the bathroom?" and "Do you have change for a $20?" might not seem to him to be valid data, and although there's no reason he couldn't decrement p using Graphite and Statsd, he'd be stuck kludging around with RRDtool. Counters like those in Coda Hale's Metrics [3] library are often used to track things such as cache entities and live threads, and for these purposes decrementors are an absolute necessity. In a more general sense, this implies that when we undertake to decide on a metrics storage system, we can no longer assume we already fully understand the primitives based on their names. We need to be sure we know what is meant when someone says "counter."

In the past few years, the field has grown laden with metrics collection systems that each have a thing called a "counter," the meaning of which is often expected to be intuitive (because, duh, what could be more obvious than a counter). But heads up. As I hope Chris and I have illustrated, counters are not simple, and they're only sometimes used for counting things.

Take it easy.

References

[1] Graphite: http://graphite.wikidot.com/.

[2] Statsd: https://github.com/etsy/statsd/.

[3] Coda Hale's Metrics: http://metrics.codahale.com/.