I guess you could say I’m between jobs at the moment. I won’t say it, because I don’t want to sound clichéd and self-conscious about being unemployed, but if you said it, it’d be fine.

You’d be right.

Don’t worry. Everything’s fine. Mostly. It’s not like I was scandalously terminated or that I rage-quit in a righteous whirlwind of well-justified sanctimony. I kind of wish it were that interesting, but no. I loved them, they loved me, it was great. And yet, filled to the brim with what can only be described as a heaping pile of privileged old dude problems, I quit.

You see, I had this plan, or maybe it was more like a nagging daydream. I couldn’t shake it. I have a little money in the bank—not really an impressive amount by my-startup-got-acquired standards, but enough to take a little time if I wanted, so I thought: “Why not just quit and drive away?”

I’d jump in my 30-year old truck and drive it north until it broke down. In that place, wherever it was, I’d talk to people who were physically standing in front of me. I’d read books made of paper, purchased from a physical store that sold books made of paper. I’d look at the clouds in the sky rather than the clouds on the other side of my VPN connection. I’d drink until my neurons realigned to real life—until character began to sound to my ears like a collection of personality traits rather than a Unicode rune, and string became a thing you tied stuff up with. I wouldn’t think about JSON, or Jinja, JVMs, or how best to organize data into structures.

I know every millimeter of exactly how stupid that sounds. They have all that stuff right where I live. Books…clouds…strings…real life. But like I said, I couldn’t shake it; like technical debt, it just seemed to keep growing, ominous and ever-present, until there was no other choice but to take a deep breath and wade in. I can hear you thinking burnout or mid-life crisis, and you’re probably right. I have no idea what I’m doing. I can say, however, that I haven’t bought a sports car, and I have no desire to write a novel, and anyway I can’t help but feel like suddenly he took a road-trip is a pretty insipid mid-life crisis, so my money is on burnout.

I’m not super worried about putting a name on it, but I became utterly convinced that indulging myself in this sad, half-baked escapist scheme would cure me. Either I’d grow back some passion for this career I’d stumbled into so many years ago, or I’d get eaten by a bear. Either outcome seemed equally likely, and I was fine with that (as long as they never caught the bear). My point is, at some point I cognitively crossed this threshold where the daydream seemed less like selfish indulgence and more like life-saving necessity.

So instead of seeing a therapist like a reasonable person, I quit (having already burned up all my vacation days and then some). Not waiting for my two-weeks’ notice to be up, I hit the road immediately. My team members were somewhat confused to suddenly find me in a Missouri coffee-shop at the next morning’s stand-up meeting, but we’re all work-from-homers anyway and my problem reports kept rolling in, so it wasn’t a huge deal. Then, as Missouri became Illinois, and Illinois became Iowa, and eventually everything became South Dakota, I feel like it became somewhat normal, if not even a little entertaining for them.
And then finally my two weeks were up, and I awoke jobless and snowed-in, in Rapid City, South Dakota, my freedom finally secured, my escape complete, my insurance revoked. I didn’t waste a single moment. I reached right into my bag, cracked open my laptop, and dug right in to Facebook’s paper on in-memory time-series databases.

Sorry, I’m new at this burnout thing. I’m sure I’ll get the hang of it eventually. On the bright side, at least I have something to share with you in this month’s column.

**Gorilla**

If you haven’t read Facebook’s paper, “Gorilla: A Fast, Scalable, In-Memory Time Series Database,” then you’re really missing out [1]. They had a problem that is extremely common in our line—er, that is to say your line—of work. Namely, too many metrics.

Having outgrown graphite, many of us—er, you—turn to OpenTSDB, the google-scale map-reduce-for-metrics system. Facebook had reached this level several years hence, and their in-house analog of OpenTSDB [2] had grown to petabyte-levels of data. Their read latency had grown in kind, such that their 90th percentile read latency was seconds long.

Facebook’s solution to this problem was to create a write-through in-memory cache system called Gorilla, which banks on a series of key observations to provide massive improvements to query-times without impacting writes.

Following that most fundamental of software engineering principles that states every problem can be solved with one additional layer of abstraction, Gorilla is inserted between the metrics-sending client nodes and Facebook’s ODS data store. Accepting posted metrics in lieu of the real persistence layer, it proxies the data to the real back end while keeping a highly compressed in-memory copy for itself. Clients can then directly query Gorilla for rapid access to recently persisted data.

One of the aforementioned observations around which Gorilla was built is that recently stored data are more valuable than older measurements. This is not surprising, but Facebook quantified it, analyzing their own query habits and discovering that 85% of their query volume targeted data less than 26 hours old.

One way Gorilla was task-optimized for its user-base is, therefore, that it only holds 26 hours worth of data. In fact, Gorilla may be the single most thoroughly spec’d out monitoring system in the history of mankind, having been specifically designed to index two billion unique time series, ingest 700 million data points per minute, and service 40,000 queries per second, to name a mere few of its many overly specific sounding design criteria. The engineers at Facebook also designed it to be horizontally scalable and resilient against their most common failure scenarios, namely, individual node failures and network partitions affecting entire regions.

There are quite a few fascinating design features in the paper, but among them, their novel approach to data compression certainly stands out.

Most metrics-oriented monitoring systems report metrics as a tuple of name (string), date (int), and value (double). Another fundamental observation the Facebook engineers made was that the timestamps in the tuples submitted to their ODS metrics system were highly periodic (data arrived on regular intervals). They therefore reasoned that rather than storing raw timestamps for every measurement in a given series, they could instead store the delta of the delta of the timestamps. For example, a hypothetically perfect time-series that reported every 60 seconds would always have a delta of 60 and delta-of-deltas of 0. By comparison, a somewhat malfunctioning time series might report at: 2:30:00, 2:31:01, and 2:31:59. These deltas would be 60, 61, and 59, and the subsequent delta-of-deltas would be 0, 1, and -1.

Writing a periodic header with a real epoch value every two hours or so would hypothetically enable you to store a much smaller numerical representation of the ongoing datetimes for a given series (0 instead of an epoch value like 1490064897). I say hypothetically because 0 actually requires len(int) bits of memory to internally represent. In other words, inside the computer, 0 is actually 0000000000, because computers are dumb, so in real life, storing 0 instead of 1490064897 doesn’t actually save you any space.

The Facebook engineers therefore eschew generic types for their own variable-length binary encoding to store these delta values. Their design works like this (where $D$ is the value of the delta-of-the-delta for a given measurement):

- If $D$ is zero, store binary 0 (only requires 1 bit of memory).
- If $D$ is between $[-63, 64)$, store ‘10’ followed by the value (7 bits).
- If $D$ is between $[-255, 256)$, store ‘110’ followed by the value (9 bits).
- If $D$ is between $[-2047, 2048)$, store ‘1110’ followed by the value (12 bits).
- Otherwise store ‘1111’ followed by $D$ using 32 bits.

Because the measurement values themselves begin life as double-precision floats rather than ints, their compression is more complex, but only slightly more so. The values are XOR’d instead of delta’d, and a similar variable-length binary encoding is employed that is based on discarding the insignificant digits of the resultant XOR’d values.

The paper reports that 96% of all inbound timestamps compress to a single-bit (i.e., stuff is mostly ‘0’) due to the periodicity of the input data (based on a random sampling of 440,000 real-world series in use at Facebook). The paper goes on to find that, for sample series that are recorded long enough (two hours seems to...
be the sweet spot), the double-precision floating point measurement values can achieve a compression ratio of 1.37 bits per data point.

Assuming 64-bit doubles, that’s 460800 uncompressed bits in a two-hour series to 9864 compressed, or a 46x compression ratio though the paper only claims a 10x compression improvement. I infer the 10x number was derived by comparing Facebook’s Gorilla implementation’s overall storage footprint to that of their HBase system.

Gorilla has also achieved the scalability, fault-tolerance, and impressive sub-millisecond read latency goals set forth by its designers, though it’s worth noting that a successful read yields compressed data (decompression is handled client-side).

Again, if you haven’t read it, it’s pretty fantastic work, and you should have a look. I mean, I read it, and I don’t even work with computers, so don’t know what you’re waiting for. It’s also worth noting that there is already some subsequent work based on Gorilla. Facebook itself has open-sourced a general-purpose reference implementation of the Gorilla daemon plus client software called Beringei [3].

Other examples include libraries that implement Gorilla’s compression algorithm, like go-tsz [4] as well as some open-source data stores like Raintank’s MetricTank [5], which uses Gorilla’s compression algorithm inside its own Cassandra-based storage back-end.

By the time you read this, I’ll hopefully still be happily unemployed—but I kind of doubt it. I’ll hold out as long as I can. Think of me when you look at the northern hemisphere.

Take it easy.

References


