Fast, real-time, granular monitoring with the tools you already have

Author: Peter Gaivoronski

Claps: 101

Date: Sep 12, 2020

As Engineers, we often find it necessary to have real-time, accurate, and granular timings of our code. These are useful for optimizing data processing, general monitoring, incident response, and many other purposes. At Klaviyo we already have a robust infrastructure for timing processes at scale via our statsd pipeline, as described by John Meichle in his earlier post. For certain real-time monitoring requirements, however, we need a different system that uses granular rather than statistical timings. Statistical systems have two downsides that make them unsuitable for measuring individual events that have a wide primary key distribution and need specific alerting.

The first downside is that the number of metrics is limited. This is a problem because for example we cannot have granular timings that include primary keys. So we cannot know how long some particular process took to execute for some particular set of data. We can only keep track of averages, percentiles, and other statistics for each metric that we create.

Another downside is that the monitoring itself only occurs when we explicitly call the monitoring function for a particular metric in the code. This becomes a problem for long running processes that need to be timed while they are running. It is possible to put a timer at every intermediate step of the process, but it becomes cumbersome to have to do that for large chunks of code. Not only do we then end up with lots of lines of code that can look like timer.time(â€˜step_7_of_28_in_process_38â€™), aggregating them on dashboards becomes a chore due to having to know every step of every process. Some steps may also end up being very long for just one line of code as well, such as a particularly heavy database insert, and therefore we still cannot time it while it is happening, since our timer now can only tell us how long the insert took after the insert happened.

When we combine these downsides, alerting in real time on a particular process stuck in the middle of an expensive query that takes tens to hundreds of minutes becomes incredibly difficult. We can only know after the fact that the process was stuck, and not in a granular way but in a statistical way that involves lots of percentile math and potentially flappy and non-specific alerting.