Profiling your code

Author: Vidhu Bhatnagar

Claps: 13

Date: Aug 11, 2020

Profiling your code, especially under load, can give you some insightful views on your codeâ€™s performance.

Last time I talked out how we load tested our events pipeline and made infrastructure changes to handle the load we estimated weâ€™d receive during Black Friday / Cyber Monday. This change was on the macro level but what can we do on the micro-level to optimize for throughput? Here Iâ€™ll talk about profiling a running process to see what areas you could improve on to troubleshoot your code or squeeze out more performance.

We recently refactored our email rendering pipeline. Towards the end of this project, we profiled our code to find ways to cut down the total render time for a given email render. During profiling, we discovered a 3rd party library that was taking longer to run than expected.

In our case, we used PyFlame to profile celery worker processes that rendered emails. PyFlame is now deprecated but any profiler could achieve the same results.

Our dataset included 15K emails and 500 mock recipients for each email. This totals 7.5 million email renders. We hooked the profiler onto our target process and profiled for 5 min.

Visualizing the data

Our favorite tool to visualize the profiling data is speedscope. Speedscope generates an interactive flame graph which allows you to visualize your profiled data with both a birdâ€™s-eye view and a detailed view of your processâ€™s call-stack and run time.

A birdâ€™s-eye view of our data looks like this: