Speaker: A. Jesse Jiryu Davis

Title: Python Performance Profiling: The Guts And The Glory

Abstract:

Setting the scene

My boss alerted me to an article on a popular site, which claimed to show that my open-source Python client for MongoDB is three times slower than the Javascript client. Anxiety immediately set in: Was this true? Could I improve it? What should I tell my boss?

Why profile?

A typical program spends almost all its time in a small subset of its code. Optimizing those hotspots is all that matters. This is what a profiler is for: it leads us straight to the functions where we should spend our effort. So I decided to apply a profiler to the code in the article to see why it was slow.

Which profiler?

I'll describe three open-source profilers for Python: cProfile is a fast single-thread profiler included in the Python standard library. GreenletProfiler is my package for profiling Gevent applications. Yappi is a third-party package that can profile multiple threads. I used Yappi for this investigation, since it's the most featureful.

How do we profile and what information do we get?

Yappi has configuration options for how it measures time and which functions it profiles. I'll show you how I configured and ran Yappi, and how I visualized its output in KCacheGrind.

How do we use the profiling information?

I used KCacheGrind's different views to narrow the search for hotspots, and calculated upper bounds for what performance enhancements I could achieve. Optimization is like debugging: we

form a hypothesis for what changes will yield the best speedups, than perform experiments. This forms a virtuous cycle of benchmarking and improving our code. I'll relate the shocking conclusion to my investigation of the slow code.

How does profiling work?

If you're like me, you can't sleep if you don't understand how something works. We'll briefly explore how cProfile and Yappi hook into the Python interpreter's guts, and how Yappi employs a clever trick to efficiently profile all running threads.