Independent Study - Tracebin Documentation

Release Spring 2012

Alex Gaynor

December 19, 2016

Contents

1 Goals **3**

2 Visualization Design **5**

3 Progress **7**
3.1 February 18, 2012 7
3.2 February 21, 2012 7

4 Future Notes **9**

Contents:

CHAPTER 1

Goals

This project has two goals:

1. To create a solid foundation for a long term performance analysis and profiling tool for PyPy.
2. To explore new ways of visualization profiling and performance information.

To make these a reality, the following goals must be met:

- The `tracebin` client must record useful information on the state of the program.
- The `tracebin` client must be able to serialize the data and send it to the server.
- The server must be able to store the data and display it (without any advanced visualization).
- The server must be able to render a page of Python code interpolated with resops and machine code.
- The server must be able to display a creative, novel, visualization of the data. What exactly this means is still up in the air. Some ideas for this are:
  - A density map of packages, modules, classes, and files based on JIT'd code.
  - An overlay of coverage data vs. trace data, showing what code is traced, but not covered by tests.
  -

These more or less need to be completed in the order described.

There is also a likelihood that new features will need to be added to the JIT hooks, or new inspiration will be needed for the UI/UX/IA of the frontend. Luckily there are two exceptionally talented friends assisting:

- Maciej Fijalkowski: PyPy core developer, and original developer of the JIT hooks.
- Idan Gazit: Brilliant designer, and inspiration for some of the current UI.

CHAPTER 2

Visualization Design

One of the major goals of this project is to develop a new visualization to better express profiling data. My goal is to take this visualization [1] and extend and improve it. One of the major advantage of this visualization is it gives not only good information about how long calls took, but provides context about the call graph and the timeline of the program. Some of the key goals are:

- See individual functions taking a large portion of the execution time.
- See smaller functions which are called repeatedly and in aggregate take a large portion of the execution time.
- Have a sense of the timeline of the program, preferably overlaid with other key timeline events, such as:
  - VM data (JIT compilation, GC minor, GC major).
  - User application metadata.

Some of the interaction mechanisms envisioned (which will need to be mapped to user input somehow):

- Zooming in on a single call, and it's sub calls.
- Graying out data:
  - All non-child calls.
  - All calls not to the same function.
  - All calls in different classes, modules, packages, etc.
- Coloring by other considerations (default coloring is "same call, same color"):
  - By call frequency (more saturated, more frequently called?)
- Reorganization of the visualization based on other events (timeline view is the default):
  - Based on total call time (one function per-line).

Some input mechanisms available:

- Hover (including a delayed hover)
- Click
- Double click
- A filter box

[1] Originally developed by Neelakanth Nadgir

CHAPTER 3

Progress

This documents both the evolution of the visualization, as well as (at least in the beginning), my learning `d3.js`.

3.1 February 18, 2012

First day with `d3.js`, using a fake dataset that contains the following calls:

Function	Start time	End time
`main`	`0`	`50`
`foo`	`1`	`8`
`bar`	`10`	`38`
`baz`	`11`	`12`
`baz`	`13`	`14`
`fizz`	`18`	`32`
`cleanup`	`39`	`45`

Image:

Basic (read: ugly) rendering of the shapes works. They're all the same color, and have no labels or interactivity.

3.2 February 21, 2012

After learning far more about color systems than I ever intended I now have working coloring for different calls. I also played with the sizing so it looks a bit more sane, added a stroke to make it look nicer, and a simple mouseover effect. As a result of all my work on learning HSV this also resulted in a patch for `d3.js` to add HSV support.

Image:

CHAPTER 4

Future Notes

Notes and links to myself for the future:

- http://en.wikipedia.org/wiki/Color_difference
- http://en.wikipedia.org/wiki/Just-noticeable_difference
- http://en.wikipedia.org/wiki/File:CIExy1931_MacAdam.png