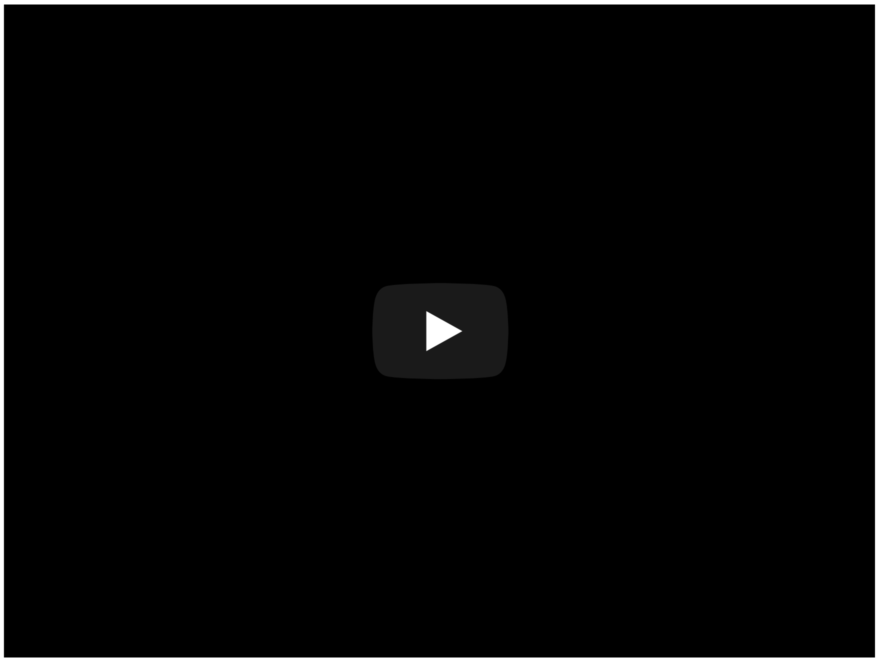
Netflix Technology Blog Follow
Learn more about how Netflix designs, builds, and operates our systems and engineering organizations
Apr 4 · 6 min read

Netflix FlameScope

We’re excited to release FlameScope: a new performance visualization tool for analyzing variance, perturbations, single-threaded execution, application startup, and other time-based issues. It has been created by the Netflix cloud performance engineering team and just released as open source, and we welcome help from others to develop the project further. (If it especially interests you, you might be interested in joining Netflix to work on it and other projects.)

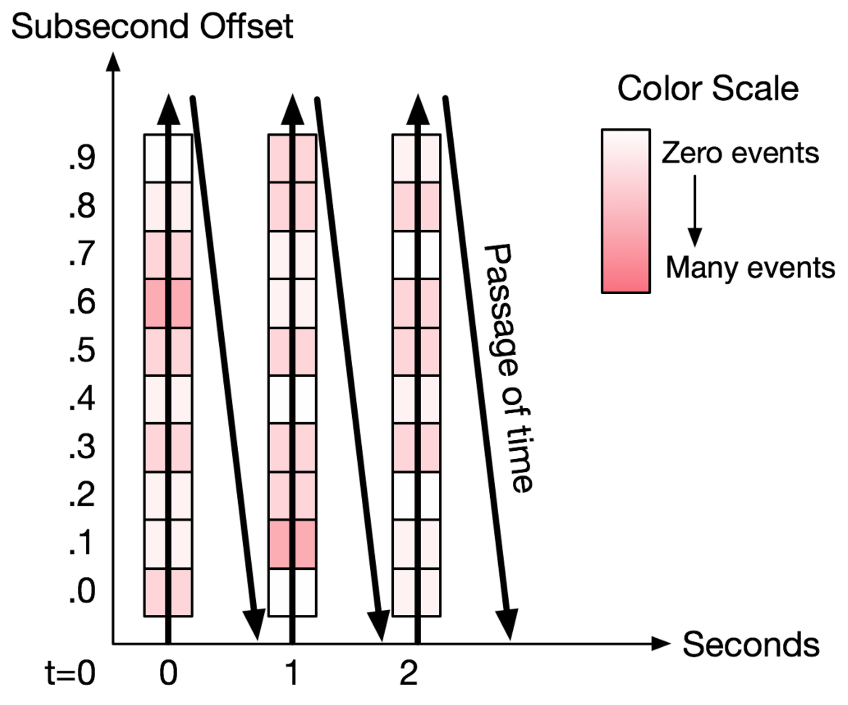
FlameScope combines a subsecond-offset heatmap for navigating a profile with flame graphs. This profile can be of CPU samples or other events. Since it’s visual, it’s best demonstrated by the following one minute video:



There is also a longer video of examples here.

Subsecond-Offset Heat Maps

If you're familiar with flame graphs, you'll know they show an entire profile at once, which can span one minute. That's good for analyzing steady workloads, but often there are small perturbations or variation during that minute that you want to know about, which become a needle-in-a-haystack search when shown with the full profile. FlameScope solves this by starting with a subsecond-offset heat map to visualize these perturbations, then lets you select them for study with a flame graph. In other words, you can select an arbitrary continuous time-slice of the captured profile, and visualize it as a flame graph.



You might not be familiar with subsecond-offset heat maps. They work as shown in figure 1, which has a mock ten row heat map, where:

- **x-axis**: the passage of time, where each column represents one second
- **y-axis**: this is also time, showing the fraction within the second: its subsecond offset
- **color**: shows how many events or samples that fell in that time range: darker for more

Imagine you have an event timestamp of 11.25 seconds. The x coordinate will be the 11th column, and the y coordinate will be the row that's one quarter from the bottom. The more events that occurred around 11.25 seconds, the darker that block will be drawn.

Example

Here's an example, with annotations showing the steps for selecting a range:

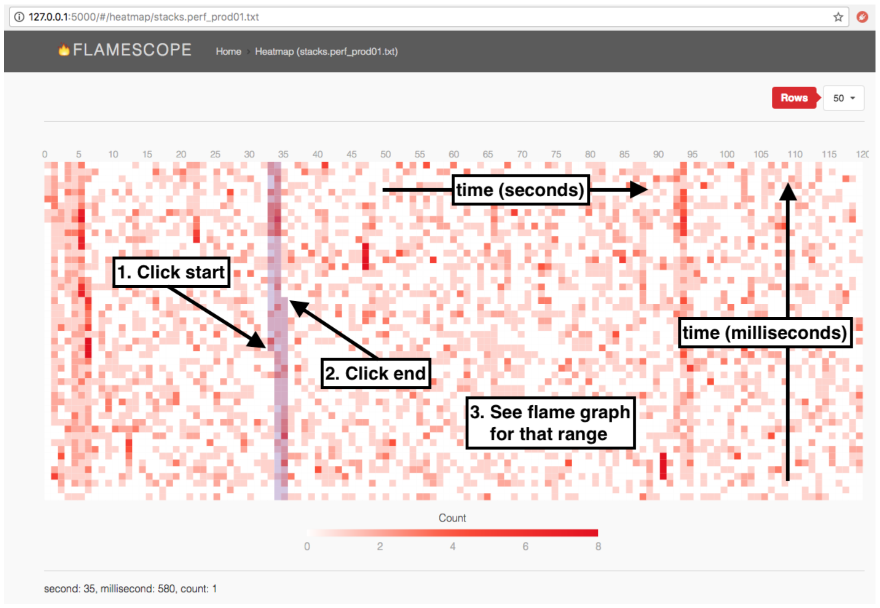


Figure 2. FlameScope selecting a time range

There's a number of interesting things from this production CPU profile. The CPUs are busier between 0 and 5 seconds, shown as darker colors. Around the 34 and 94 second mark (sounds like a 60 second periodic task), the CPUs also become busier, but for a shorter duration. And there are occasional bursts of heavy CPU activity for about 80 milliseconds, shown as short dark red stripes.

All of these details can be selected in FlameScope, which will then draw a flame graph just for that range. Here's one of the short red stripes:

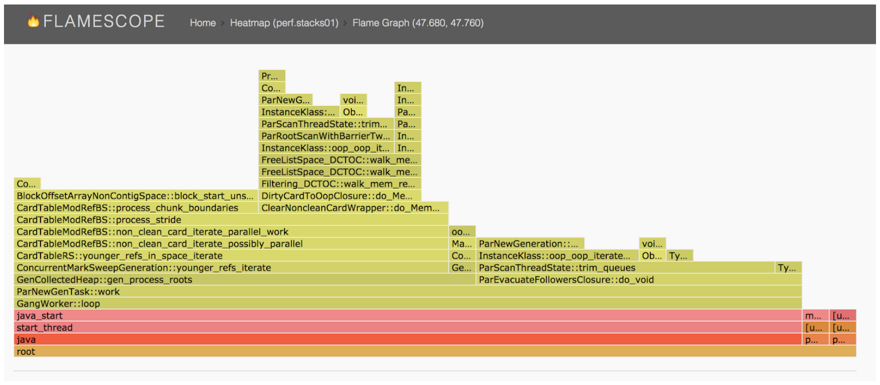

FLAMESCOPE
Home
Heatmap (perf.stacks01)
Flame Graph (47.680, 47.760)
Pr...
Co...
ParNewG...
voi...
In...
In...
InstanceKlass:...
Ob...
Pa...
ParScanThreadState::trim...
Pa...
ParRootScanWithBarrierTw...
In...
InstanceKlass::oop_oop_it...
In...
FreeListSpace_DCTOC::walk_me...
FreeListSpace_DCTOC::walk_me...
Co...
Filtering_DCTOC::walk_mem_re...
BlockOffsetArrayNonContigSpace::block_start_uns...
DirtyCardToOopClosure::do_Me...
CardTableModRefBS::process_chunk_boundaries
ClearNoncleanCardWrapper::do_Mem...
CardTableModRefBS::process_stride
CardTableModRefBS::non_clean_card_iterate_parallel_work
oo...
CardTableModRefBS::non_clean_card_iterate_possibly_parallel
Ma...
ParNewGeneration::...
voi...
CardTableRS::younger_refs_in_space_iterate
Co...
InstanceKlass::oop_oop_iterate...
Ob...
Ty...
ConcurrentMarkSweepGeneration::younger_refs_iterate
Ge...
ParScanThreadState::trim_queues
Ty...
GenCollectedHeap::gen_process_roots
ParEvacuateFollowersClosure::do_void
ParNewGenTask::work
GangWorker::loop
java_start
m...
[u...
start_thread
[u...
[u...
java
p...
p...
root

Figure 3. Flame graph for a selected time range

Ah, that's Java garbage collection.

Instructions

Getting started instructions are listed (and will be updated) on the github repository here. The quickest way to get started is:

```
$ git clone https://github.com/Netflix/flamescope
$ cd flamescope
$ pip install -r requirements.txt
$ python run.py
```

FlameScope comes with a sample profile to browse (where application code has been redacted with 'x' characters). Here's how to create new profiles on Linux, which can be added to the examples directory of FlameScope for browsing:

```
$ sudo perf record -F 49 -a -g -- sleep 120
$ sudo perf script --header >
stacks.myproductionapp.2018_03_30_01
$ gzip stacks.myproductionapp.2018_03_30_01    # optional
```

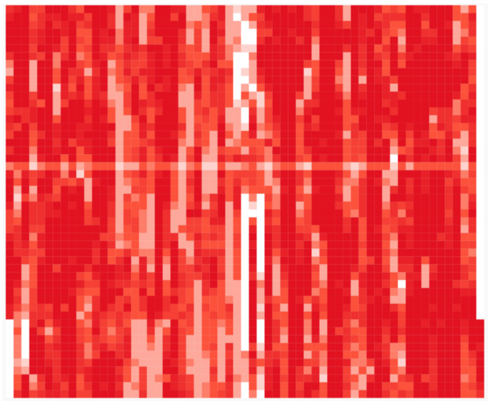
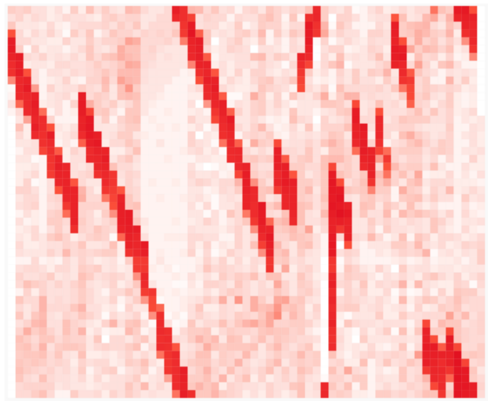
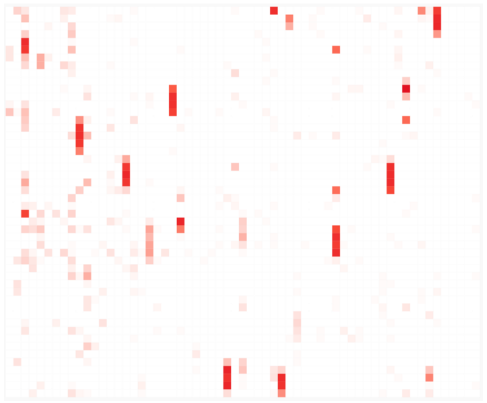
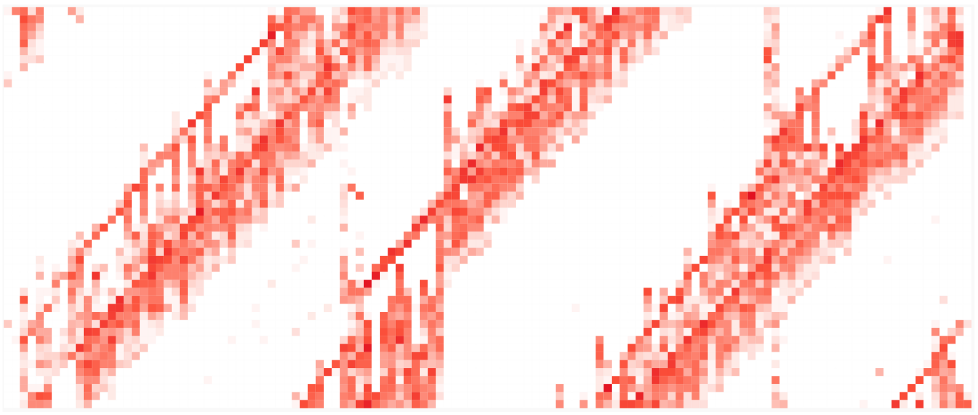
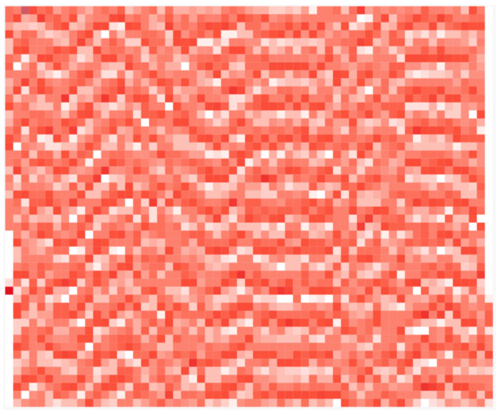
That example shows a two minute CPU profile, sampling at 49 Hertz on all CPUs. Any perf output with stack traces can be browsed with FlameScope, including tracing events such as block I/O, context switches, page faults, etc. Since the profile output can get large, it can also be compressed with gzip (flamescope can read .gz).

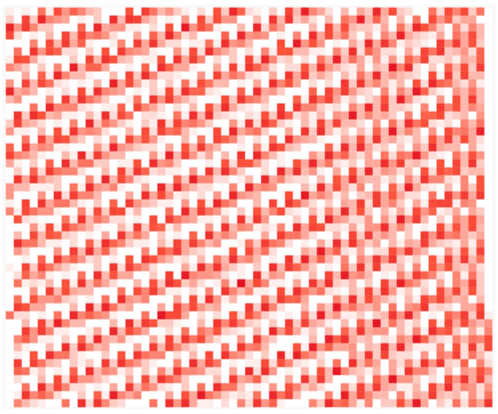
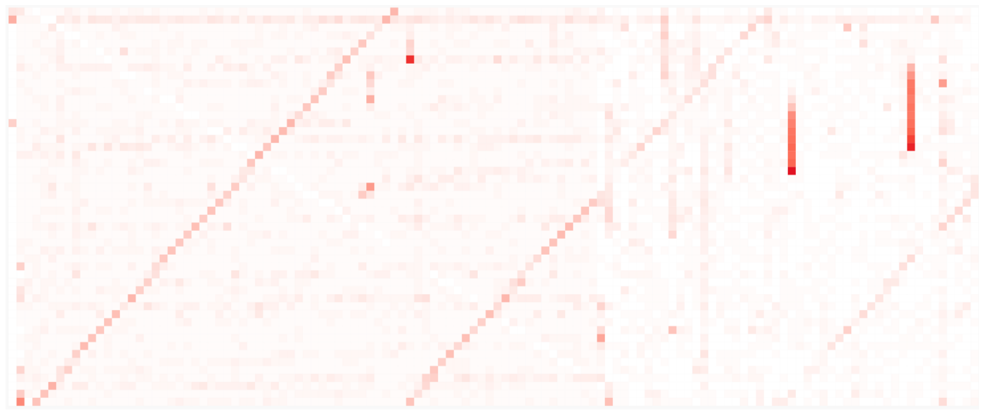
Why sample at 49 Hertz? Because 50 Hertz may sample in lock-step with a timed activity, and over- or under-count. Why roughly 50 Hertz in the first place? It's not too slow and not too fast. Too slow and we don't have enough samples to paint across FlameScope's 50 row heatmap (the row count can be changed). Too fast and the overhead of sampling can slow down the application.

Runtimes like Java can require extra steps to profile using perf correctly, which have been documented in the past for generating flame graphs (including here). Since you may have already been running these steps, you might have a library of old profiles (perf script output) that you can now explore using FlameScope.

Screenshots

Since FlameScope reads Linux perf profiles, I already have a collection from prior investigations. Here are some screenshots, showing variation that I did not know about at the time.

Not all profiles are this interesting. Some do just look like TV static: a steady workload of random request arrivals and consistent latency. You can find out with FlameScope.

Origin

FlameScope was created by the Netflix cloud performance team, so far involving Vadim Filanovsky, myself, Martin Spier, and our manager, Ed Hunter, who has supported the project. The original issue was a microservice that was suffering latency spikes every 15 minutes or so, cause unknown. Vadim found it corresponded to an increase in CPU utilization that lasted only a few seconds. He had tried collecting CPU flame graphs to explain this further, but a) could not reliably capture a one minute flame graph covering the issue, as the onset of it kept fluctuating; and b) capturing a two or three minute flamegraph didn't help either, and the issue was "drowned" in the normal workload profile. Vadim asked me for help.

Since I had a two minute profile to begin with, I began by slicing it into ten second ranges, and creating a flame graph for each. This approach looked promising as it revealed variation, so I sliced it even further down to one second windows. Browsing these short windows solved the problem and found the issue, however, it had become a laborious task. I wanted a quicker way.

Subsecond-offset heat maps were something I invented many years ago, but they haven't seen much adoption or use so far. I realized they would be a great way to navigate a profile, allowing variance to be visualized not just for whole seconds but also for fractions within a second. I did a quick prototype which proved the idea worked, and discussed turning it into a real tool with Martin. The annotated heat map in this post shows Vadim's original profile, and the issue was the CPU activity in the first few seconds.

Martin has done most of the architecture design and coding for FlameScope, which includes his newer d3-based version of FlameGraphs: d3-flame-graph, and his d3-based heat maps: d3-heatmap2. It's great to see them come together in FlameScope.

Future Work

There's more features we have planned, and we'll add tickets to github in case others would like to help. They include more interactive features, such as palette selection and data transformations. There should be a button to save the final flame graph as a stand alone SVG. Other profile sources can be supported, not just Linux perf. And there should be a way to show the difference between the selected range and the baseline (the whole profile).

If you're reading this months in the future, some of these extra features may already exist: check out the latest on https://github.com/Netflix/flamescope.

FlameScope was developed by Martin Spier and Brendan Gregg, Netflix cloud performance engineering team. Blog post by Brendan Gregg.