

Causality based Profiling

Bernhard Schuster

December 1, 2020

~~Causality-Based~~ Profiling

Motivation

Profiling - Motivation

- Time from start to end of operation
- Resource usage
- Resource utilization
- Items processed per time unit
- QoS
- Snappy-ness
- ..

Profiling yields

Measures time spent in scopes



choices

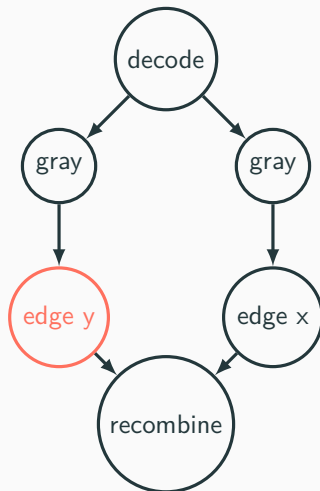
- <https://lib.rs/crates/flamegraph>

- <https://lib.rs/crates/flamegraph>
- <https://lib.rs/crates/puffin>

- <https://lib.rs/crates/flamegraph>
- <https://lib.rs/crates/puffin>
- <https://lib.rs/crates/tracing>

- <https://lib.rs/crates/flamegraph>
- <https://lib.rs/crates/puffin>
- <https://lib.rs/crates/tracing>
- <https://lib.rs/crates/mick-jaeger>

Profiling - Demo App Architecture





flamegraph

List of spent time per scope.

Profiling - Flamegraph

List of spent time per scope.

Hint: Try to keep an eye on the threads



Demo: flamegraph

How does that help us?

What's our goal?

Optimize

Optimize

- Throughput
- Latency

Optimize


- Throughput
- Latency

of the end to end execution

Optimize

- Throughput
- Latency

of the **end to end** execution in heavily **threaded** applications.

A photograph of a roller coaster track against a cloudy sky. The track is dark grey with white support pillars. Two trains are visible: one at the peak of a large loop and another on a downward slope. A dark green rectangular box with white text is centered over the middle of the image.

Influence of Locking

A photograph of a roller coaster track with two cars visible. The track is made of dark metal and features several loops and drops. The cars are filled with passengers. The background is a cloudy sky. A dark horizontal bar with white text is overlaid on the center of the image.

Component/Scope vs System/Service

Causality Based Profiling

Causality Based Profiling

coz - throughput

```
// Throughput
coz::scope!("foo");

// equiv to:

coz::begin!("foo");
// ..
coz::end!("foo");
```

```
// Latency  
coz::progress!("foo");
```



Demo: coz

How it works - semantics

- Performance measurements
- Delay injection for all threads other than the one running the $f(x)$
- Randomized delays

How it works - mechanics outline

- LD_PRELOAD
- Intercept posix API
- Counter and delay tracking per thread

- <https://www.youtube.com/watch?v=jE0V-p1odPg>
- <https://github.com/plasma-umass/coz/>
- <https://arxiv.org/pdf/1608.03676v1.pdf>

Best use cases

- Component ∞ System/Service
- Easily repeatable
- Representative input data distribution
- Determine best investment for the given system

Caveats

- Sampling data points take up to a few hours
- OOM Conditions skew the experiments
- Bezier interpolation of plot representation sometimes misleading

Questions?

bernhard+rustmeetup@ahoi.io

Credits

Presentation by Bernhard Schuster - ahoi.io

Theme by Matze Vogelsang and contributors -

<https://bloerg.net>

The theme *itself* is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The images are taken from *unsplash* and are free to use for whatever you want.

Font Awesome icons SIL OFL 1.1

Icons displayed made by Bernhard Schuster.

