Tracing

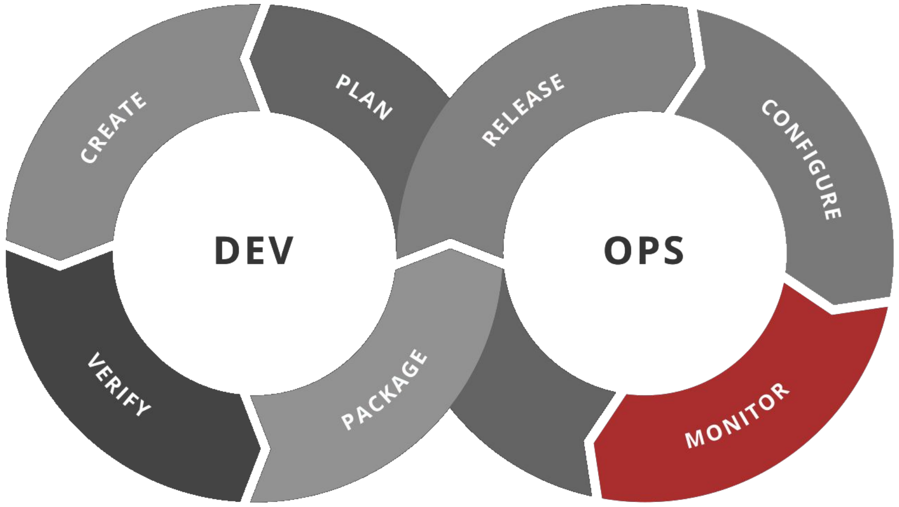
For performance optimization in distributed systems

Johan Vikström

Sandro Lockwall Rhodin

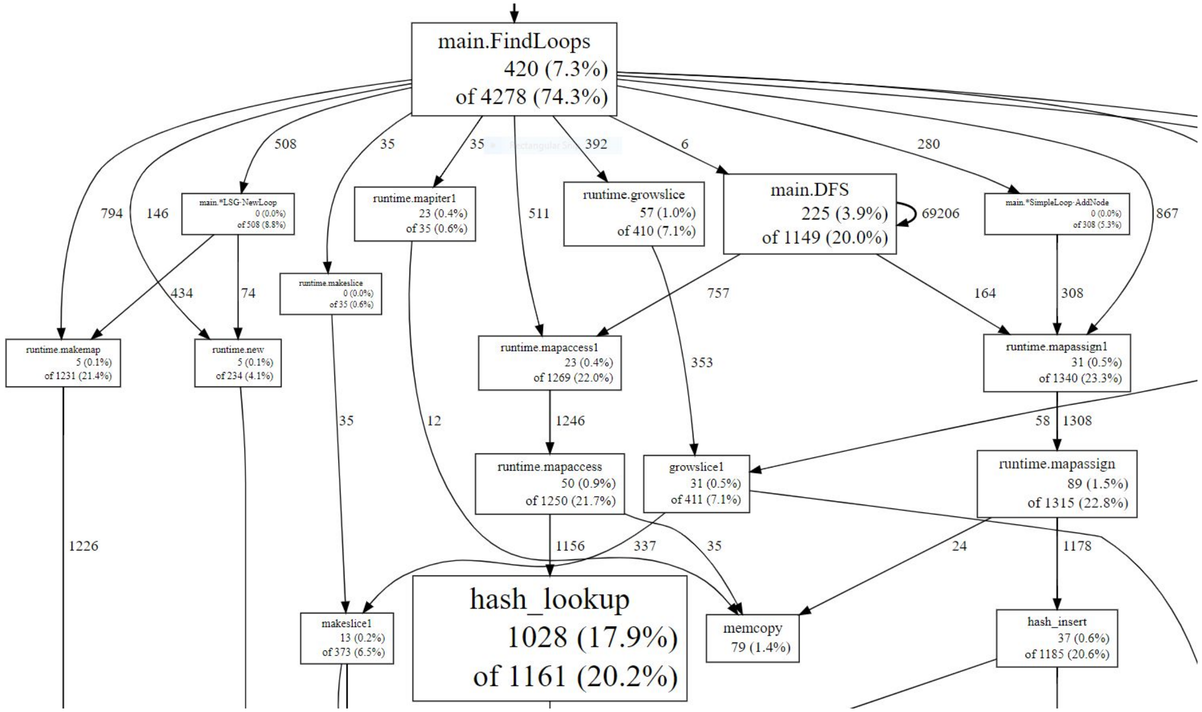
Relation to DevOps

- Found within the Monitor aspect of the DevOps loop
- Often used together with other observability tools (such as metrics based ones)
- Usefulness in providing a better understanding of applications built on microservices



Performance optimization...

- Performance optimization on a single host is “easy”
  - Just run “perf”, “pprof”
     or “Callgrind”
    and get a callgraph
  - Use to find hotspots to
    optimize

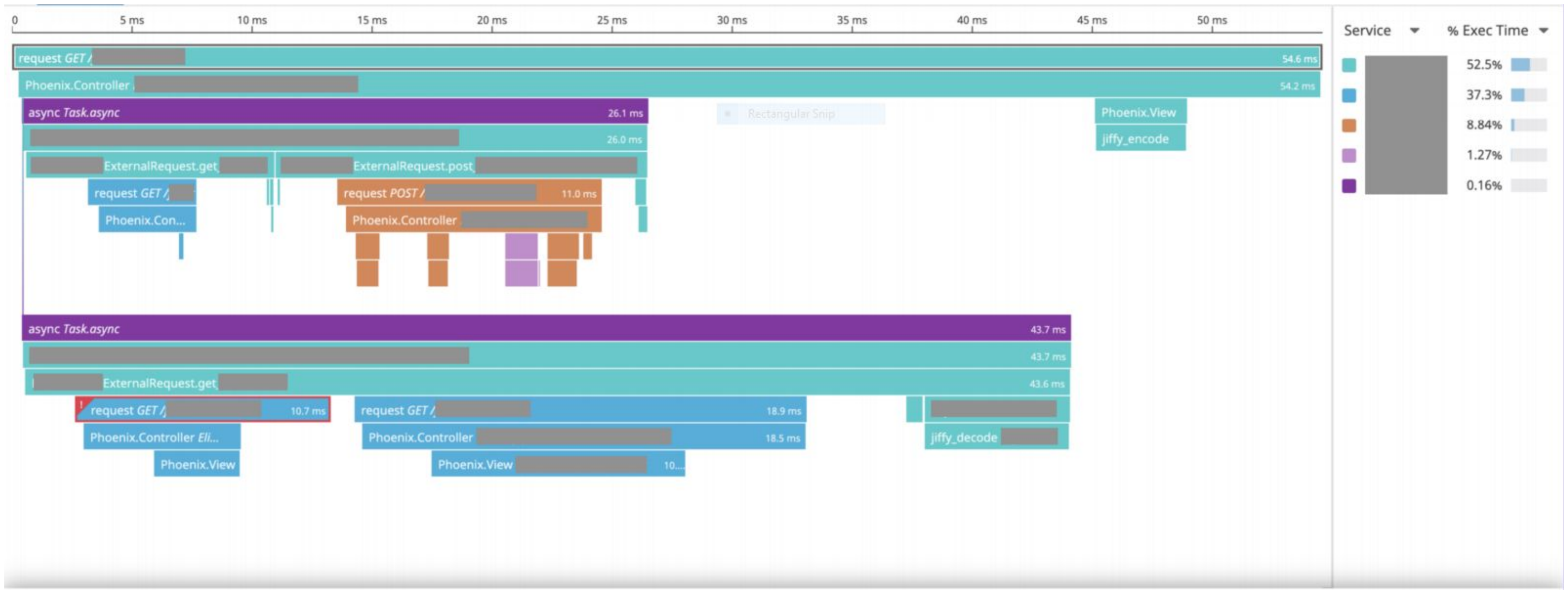


… In a distributed setting...

- Not only CPU usage impacts performance
- Also must account for latency between services
- High CPU usage $\not\Rightarrow$ “Bad performance”
    - Maybe other tasks run in parallel?
    - Causal relationships are much more important
- So optimizing using Callgraphs and CPU usage is a no go

....Tracing to the rescue!

Measure latency between each service, keep track of relationships and generate flame graphs



https://github.com/GregMefford/presentations/releases/tag/2019-code-beam-sto

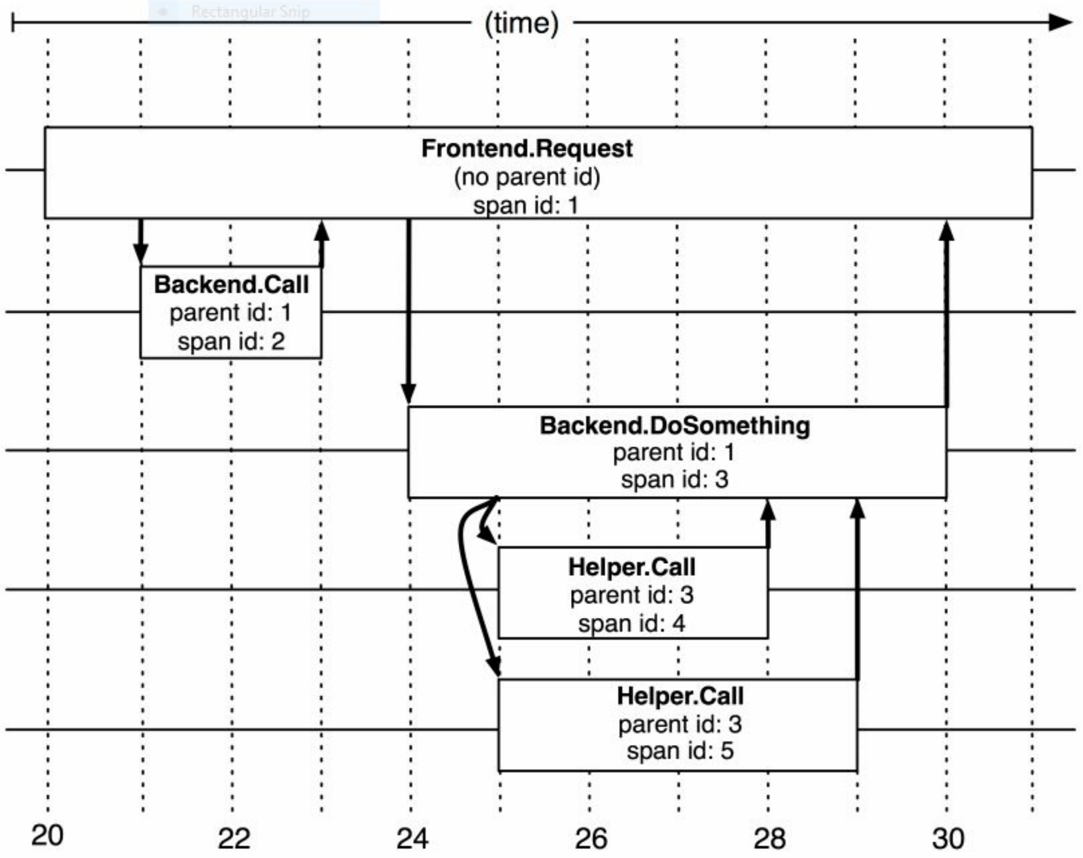
How is this done?

Metadata propagation! (Dapper)

- Trace ID
- Span ID
- Parent Span ID

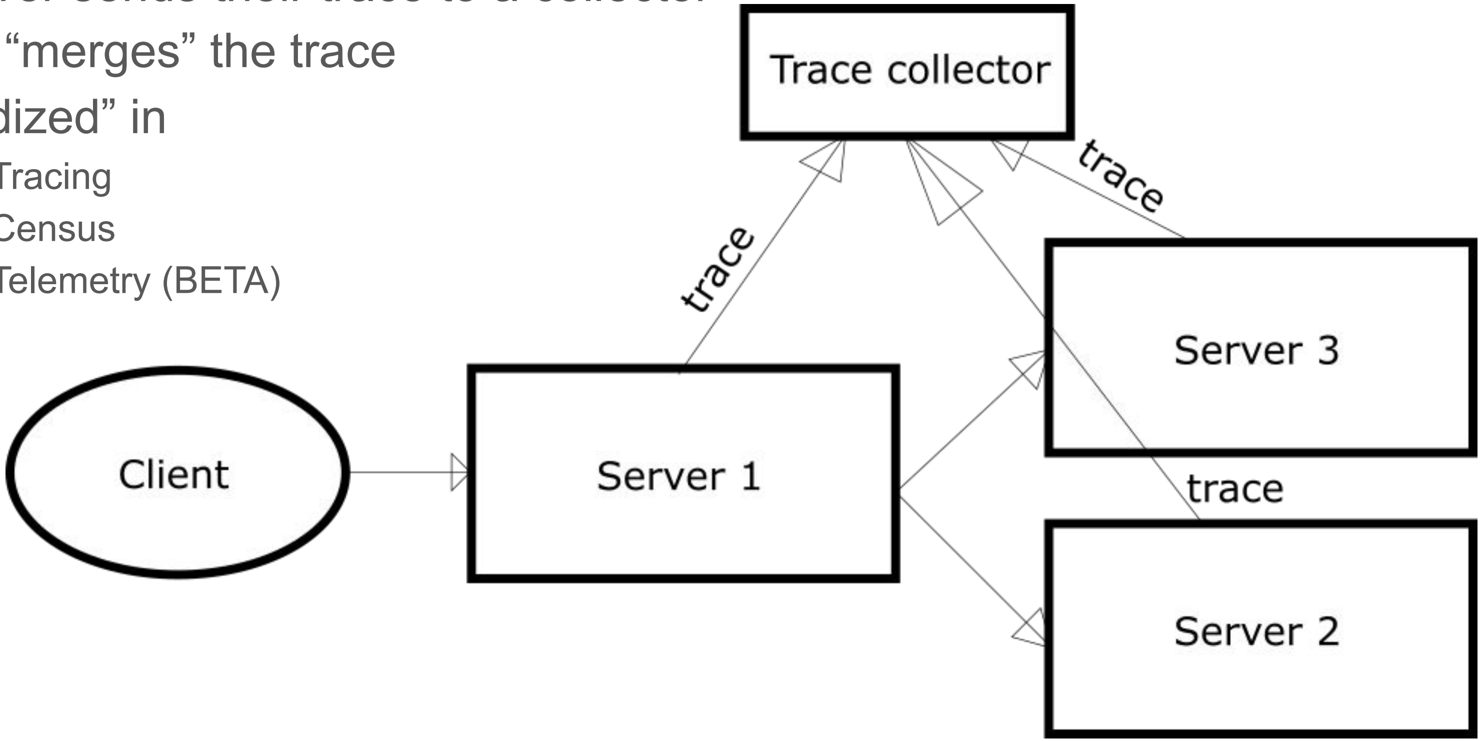
- Propagate via Headers
  - Inject into RPCs
- Send to trace collector

https://research.google/pubs/pub36356/

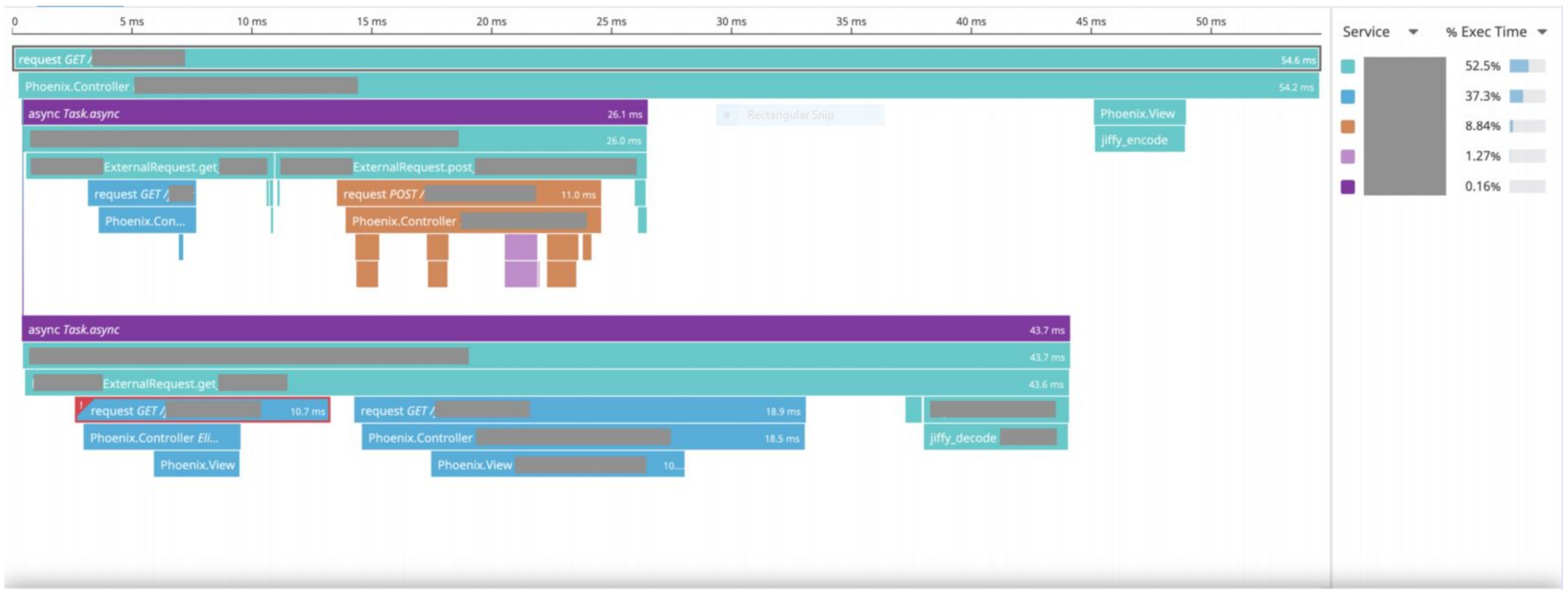


How is this done?

- Each server sends their trace to a collector
- Collector "merges" the trace
- "Standardized" in
    - OpenTracing
    - OpenCensus
    - OpenTelemetry (BETA)



What do we do with the trace?



Existing Solutions to the Tracing Problem

- There are a number of software solutions, Jaeger, Zipkin, Datadog
- They simplify the amount of work that has to be done by the individual in need of tracing.
- There are some pros and cons to using each of these different services.

Official Support	C++	C#	Go	Java	Node.js	PHP	Python	Ruby	Scala
Jaeger	Yes	Yes	Yes	Yes	Yes	**No**	Yes	**No**	**No**
Zipkin	**No**	Yes	Yes	Yes	Yes	Yes	**No**	Yes	Yes

Real-life Example: Uber

- Uber utilises about 2000 microservices to run their applications.
- Uber has used a number of different tracing tools to try to solve their problems
- Uber made their own tracing tool, Jaeger
- Tracing allows them to quickly identify how something has gone wrong.

So why should you be tracing?

If you are part of essentially any modern microservice project, tracing is an essential tool to understand how to improve upon your projects performance.

If you’re not tracing, you’ll be spending more time improving things that don’t matter in the long run.

Don’t **waste** your time, **trace instead**, or your application might end up **dead**.

Sources

Tracing:

https://medium.com/opentracing/take-opentracing-for-a-hotrod-ride-f6e3141f7941
https://www.jaegertracing.io/docs/1.17/frontend-ui/
https://gianarb.it/blog/faq-distributed-tracing
https://opentracing.io/docs/overview/spans/
https://opentracing.io/docs/overview/what-is-tracing/
https://medium.com/homeaway-tech-blog/request-tracing-for-fun-and-profit-31bd5beb58c0
https://opensource.com/article/18/9/distributed-tracing-microservices-world
https://petabridge.com/blog/why-use-distributed-tracing/
https://www.infoq.com/articles/distributed-tracing-microservices/

Existing Solutions:

https://logz.io/blog/zipkin-vs-jaeger/
https://zipkin.io/pages/tracers_instrumentation
https://www.jaegertracing.io/docs/1.17/client-libraries/
https://opentracing.io/docs/supported-tracers
https://www.datadoghq.com/product/

Uber:

https://eng.uber.com/distributed-tracing/
https://www.infoq.com/presentations/uber-microservices-distributed-tracing/

Google:

Sigelman, Benjamin H., et al. "Dapper, a large-scale distributed systems tracing infrastructure." (2010).