Tracing polyglot systems
An OpenTracing Tutorial

Yuri Shkuro (Uber), Won Jun Jang (Uber), Prithvi Raj (Uber)

Velocity NYC, Oct 1 2018

- 9:00 - 9:15 Introductions
- 9:15 - 9:45 (talk) **Introduction to Distributed Tracing**
- 9:45 - 10:00 Q & A
- 10:00 - 10:30 **Tutorials**
- 10:30 - 11:00 Break
- 11:00 - 11:15 Part 2: Q & A
  - How far did you get?
  - Any questions about the OpenTracing API?
- 11:15 - 11:45 Tutorials (continued)
- 11:45 - 12:00 (talk) **Deploying and Using Tracing in Your Organization**
- 12:00 - 12:30 Group discussion / unconference
Getting the most of this workshop

- Learn the ropes.
- If you already know them, help teach ‘em ropes :) 
- Meet some people

Everyone can walk away with practical tracing experience and a better sense of the space.
Intros

● Which company / organization are you from?
● How big is your architecture?
● What monitoring challenges do you have?
Why care about Tracing

Tracing is fun
Modern applications are very complex.

Thanks, microservices!
BILLIONS times a day!
How do we know what’s going on?
We use MONITORING tools

**Metrics / Stats**
- Counters, timers, gauges, histograms
- Four golden signals
  - utilization
  - saturation
  - throughput
  - errors
- Statsd, Prometheus, Grafana

**Logging**
- Application events
- Errors, stack traces
- ELK, Splunk, Fluentd

Monitoring tools must “tell stories” about your system
Metrics and logs don’t cut it anymore!

Metrics and logs are per-instance. They don’t tell the full story.

We need to understand distributed transactions.
Systems are **Distributed and Concurrent**

“The Simple [Inefficient] Thing”

- **Basic Concurrency**
- **Async Concurrency**

**Distributed Concurrency**
How do we “tell stories” about distributed concurrency?
Accela gets acquired by Berkshire Partners as it looks to move govttech services to the cloud

by John Mannees

Boston-based private equity shop Berkshire Partners announced this afternoon that it is acquiring Accela — a nearly 20 year old startup that sells regulatory management solutions to government clients. Accela has gone through a troika of CEOs in the last year. Previously acting CEO Mark Jung replaced Maury Blackman last October who had managed the company for about a decade. We... Read More
Distributed Tracing in a Nutshell

Unique ID → {context}

Edge service

TRACE

SPANS
Let’s look at some traces

demo time: http://bit.do/jaeger-hotrod
Distributed Tracing Systems

- distributed transaction monitoring
- root cause analysis
- distributed context propagation
- performance and latency optimization
- service dependency analysis
Great… Why isn’t everyone tracing?

Tracing instrumentation has been too hard, with no standardization.
### How are applications instrumented?

<table>
<thead>
<tr>
<th>Application (manually instrumented)</th>
<th>Application (automatically instrumented)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually instrumented frameworks</td>
<td>Agent for automatic instrumentation</td>
</tr>
</tbody>
</table>

- **Open Source Instrumentation API**
- **Tracing library implementation**
- **Tracing system / analytics backend**
A Bigger Picture

Your Service

- Shared Libraries
- Tracing API
- Tracer

Describing Transactions

Recording Transactions

Correlating Transactions

Federating Transactions

Not Your Service (Spanner, S3, Kinesis, etc.)

Not Your Tracing System (StackDriver, XRay)

OPEN TRACING

Trace Data

Analyzing Transactions

Your Tracing System (Jaeger, Zipkin)

Trace Data

Trace-Data
What is OpenTracing

http://opentracing.io
OpenTracing Mission

Provide an API for describing distributed transactions

Unlock open source, vendor-neutral instrumentation
OpenTracing Goals

- Zero-dependencies, **pure API for describing** the shape, timing, and metadata about **distributed transactions**. Vendor neutral. Data formats agnostic.

- **API primitives for** intra-process and inter-process **propagation of context**, including general purpose, transaction-scoped “baggage”.

- **A body of** reusable, vendor-neutral, **open source instrumentation** for existing systems, libraries, and frameworks, and/or enable them to include instrumentation built-in.

- **Semantic conventions** for standardized data elements (for tags and log fields) for describing metadata of common operations, such as http or database calls.
Who should care?

Developers building:

- Cloud-native / microservices-based applications
- OSS packages, especially near process edges (web frameworks, managed service clients, etc)
- Tracing and/or monitoring systems
OpenTracing Architecture

- microservice process
  - application logic
  - μ-service frameworks
  - Lambda functions
  - RPC & control-flow frameworks
  - existing instrumentation

OpenTracing API

main()

CNCF Jaeger

LIGHTSTEP

ZIPKIN

appdash

INSTANA

tracing infrastructure
A young, growing project

2.5 years old (https://opentracing.devstats.cncf.io)

Tracer implementations: Jaeger, Zipkin, LightStep, SkyWalking, others

All sorts of companies use OpenTracing:
Rapidly growing OSS and vendor support
Jaeger

A distributed tracing system
Jaeger - /ˈyāɡər/, noun: hunter

- Inspired by Google’s Dapper and OpenZipkin
- Started at Uber in August 2015
- Open sourced in April 2017
- Official CNCF project since Sep 2017
- Built-in OpenTracing support
- [https://jaegertracing.io](https://jaegertracing.io)
Jaeger Technology Stack

- Backend components in Go
- Pluggable storage
  - Cassandra, Elasticsearch, memory, ...
- Web UI in React/Javascript
- OpenTracing instrumentation libraries
Jaeger: Community

- Several full time engineers at Uber and Red Hat
- Over 600 contributors on GitHub (stats)

Blog: [https://medium.com/jaegertracing](https://medium.com/jaegertracing)
Chat: [https://gitter.im/jaegertracing/Lobby](https://gitter.im/jaegertracing/Lobby)
Twitter: [https://twitter.com/JaegerTracing](https://twitter.com/JaegerTracing)
OpenTracing deep dive

Doc http://bit.do/velocity18
Materials

- Q&A: [https://gitter.im/opentracing/workshop](https://gitter.im/opentracing/workshop)
Lesson 1
Hello, World
Lesson 1 Objectives

- Basic concepts
- Instantiate a Tracer
- Create a simple trace
- Annotate the trace
Basic concepts: SPAN

**Span**: a basic unit of work, timing, and causality.

A span contains:

- operation name
- start / finish timestamps
- tags and logs
- references to other spans
Basic concepts: TRACE

**Trace**: a directed acyclic graph (DAG) of spans

- Span A
  - Span B
    - Span D
  - Span C
    - Span E
    - Span F
  - Span G
  - Span H
Basic concepts: OPERATION NAME

A human-readable string which concisely represents the work of the span.

- E.g. an RPC method name, a function name, or the name of a subtask or stage within a larger computation
- Can be set at span creation or later
- Should be low cardinality, aggregatable, identifying *class of spans*

<table>
<thead>
<tr>
<th>get</th>
<th>too general</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_account/12345</td>
<td>too specific</td>
</tr>
<tr>
<td>get_account</td>
<td>good, “12345” could be a tag</td>
</tr>
</tbody>
</table>
Basic concepts: TAG

A key-value pair that describes the span overall.

Examples:

- http.url = “http://google.com”
- http.status_code = 200
- peer.service = “mysql”
- db.statement = “select * from users”

https://github.com/opentracing/specification/blob/master/semantic_conventions.md
Describes an event at a point in time during the span lifetime.

- OpenTracing supports **structured logging**
- Contains a timestamp and a set of fields

```python
span.log_kv(
    {'event': 'open_conn', 'port': 433}
)
```
A tracer is a **concrete** implementation of the OpenTracing API.

```go
tracer := jaeger.New("hello-world")
span := tracer.StartSpan("say-hello")
// do the work
span.Finish()
```
Understanding Sampling

- Tracing data > than business traffic
- Most tracing systems sample transactions
- **Head-based sampling**: the sampling decision is made just before the trace is started, and it is respected by all nodes in the graph
- **Tail-based sampling**: the sampling decision is made after the trace is completed / collected
How to create Jaeger Tracer

cfg := &config.Configuration{
    Sampler: &config.SamplerConfig{
        Type: "const",
        Param: 1,
    },
    Reporter: &config.ReporterConfig{LogSpans: true},
}

tracer, closer, err := cfg.New(serviceName)
Lesson 2
Context and Tracing Functions
Lesson 2 Objectives

- Trace individual functions
- Combine multiple spans into a single trace
- Propagate the in-process context
How do we build a DAG?

```go
span1 := tracer.StartSpan("say-hello")
// do the work
span1.Finish()

span2 := tracer.StartSpan("format-string")
// do the work
span2.Finish()
```

This just creates two independent traces!
Build a DAG with Span References

```javascript
span1 := tracer.StartSpan("say-hello")
// do the work
span1.Finish()

span2 := tracer.StartSpan("format-string",
    opentracing.ChildOf(span1.Context()),
)
// do the work
span2.Finish()
```
Basic concepts: SPAN CONTEXT

Serializable format for linking spans across network boundaries.

Carries trace/span identity and baggage.

type SpanContext struct {
    traceID   TraceID
    spanID    SpanID
    parentID  SpanID
    flags     byte
    baggage   map[string]string
}

CLOUD NATIVE COMPUTING FOUNDATION
Basic concepts: SPAN REFERENCE

Describes causal relationship to another span.

type Reference struct {
    Type    opentracing.SpanReferenceType
    Context SpanContext
}
Types of Span References

**ChildOf**: referenced span is an ancestor that depends on the results of the current span. E.g. RPC call, database call, local function.

**FollowsFrom**: referenced span is an ancestor that does not depend on the results of the current span. E.g. async fire-n-forget cache write.
We don’t want to keep passing Spans around.

Need a more general request context:

- Go: context.Context (from std lib)
- Java, Python: Scope & Scope Manager (thread-locals)
- Node.js: TBD (internally: @uber/node-context)
Lesson 3
Tracing RPC Requests
Lesson 3 Objectives

- Trace a transaction across more than one microservice
- Pass the context between processes using **Inject** and **Extract**
- Apply OpenTracing-recommended tags
Anatomy of Tracing Instrumentation

- **MY SERVICE**

  - **Context Span**
  - **Send trace data to tracing backend (background thread)**

- **Headers**
  - TraceID

- **inbound request**
  - **Handler**
  - **instrumentation**

- **outbound request**
  - **Client**
  - **instrumentation**

- **Tracer library**
Basic concepts: Inject and Extract

Tracer methods used to serialize Span Context to or from RPC requests (or other network comms)

```c
void Inject(SpanContext, Format, Carrier)
SpanContext Extract(Format, Carrier)
```
Basic concepts: Propagation Format

OpenTracing does not define the wire format.

It assumes that the frameworks for network comms allow passing the context (request metadata) as one of these (the Format enum):

1. **TextMap**: Arbitrary string key/value headers
2. **Binary**: A binary blob
3. **HTTPHeaders**: as a special case of #1
Basic concepts: Carrier

Each Format defines a corresponding Carrier interface that the Tracer uses to read/write the span context.

The instrumentation implements the Carrier interface as an adapter around their custom types.
Inject Example

Set(key, value)

Tracer

Write(byte[])

TextMap Carrier → RPC Adapter → AddHeader(key, value) → RPC Request

Binary Carrier → Adapter → Write(byte[]) → RPC Request
Lesson 4
Baggage
Lesson 4 Objectives

- Understand distributed context propagation
- Use baggage to pass data through the call graph
Distributed Context Propagation

Problem: how to aggregate disk writes in Cassandra by "button" type (or experiment id, etc, etc)?

See the Pivot Tracing paper [http://pivottracing.io/](http://pivottracing.io/)
Basic concepts: Baggage

Baggage is a general purpose in-band key-value store.

```
span.SetBaggageItem("Bender", "Rodriguez")
```

Transparent to most services.

Powerful but dangerous
- Bloats the request size
Extra Credit
Logging v. Tracing
Monitoring == Observing Events

- **Metrics** - Record events as *aggregates* (e.g. counters)
- **Tracing** - Record *transaction-scoped* events
- **Logging** - Record *unique* events
Logging v. Tracing

Logging

- No context
- Low granularity (warn and ↑)
- Per-process sampling (at best)
- High volume, low fidelity

Tracing

- Contextual
- High granularity (debug and ↓)
- Per-transaction sampling
- Lower volume, higher fidelity

Industry advice: don’t log on success (https://vimeo.com/221066726)
Deploying and Using Tracing in Your Organization
Practitioner’s Advice
Getting the Basics

You will need to:

1. Identify relevant frameworks
2. Install relevant OpenTracing plugins
3. Instrument code with OpenTracing
4. Pick a Tracer that matches your tracing backend
Frameworks & Plugins

Modern systems are a mix of your application code, shared libraries, and shared infrastructure/resources.

The `opentracing-contrib` project adds OT support to popular libraries and frameworks.

Leveraging these plugins both expands OT coverage in your app and may reduce the required explicit code instrumentation.
Integrate with your infra libraries early

- Create adapters, do not use outside libraries directly
  - Adapters allow you to customize configuration
  - E.g. do not expect app developers to give your tracer the service name, get it from some environment variable instead
- If your org is already using common infra libraries, e.g. for RPC, change them to include tracing by default
- Tag logs with trace & span ID, people love it
Decide What to Instrument

• Identify a high-value business transaction
  – E.g. “discover nearby x”, “add to cart”, etc.
• Identify the points of ingress and egress
• Breadth-first, not depth-first
• Get the first end-to-end trace reported
Evangelize

• Give internal talks
• Show people examples where tracing helps
• Get management buy-in
Distributed Context Propagation

- Identifying synthetic traffic
  - Use as a dimension for metrics
- Product Tenancy
  - E.g. top-level product: Docs, Gmail
- Chaos engineering & fault injection
  - Random killings must stop!
Use Data Mining

• Tracing data is extremely rich. Don’t let it go to waste by looking at individual traces only.
• Create big data jobs to aggregate traces for meaningful insights, specific to your infrastructure
  – We write spans to Kafka and run Flink jobs
  – We also support Hive queries on HDFS
In conclusion

Wow, what a great audience!
Contributors are most welcome

http://opentracing.io

http://jaegertracing.io
Thank you & Happy Tracing!

• Hope to see you in Shanghai or Seattle!
  – Registration & Sponsorships now open: kubecon.io
• KubeCon + CloudNativeCon China 2018
  – 11月13-15日 November 13 – 15, 2018 | 中国上海 Shanghai, China
• KubeCon + CloudNativeCon North America 2018
  – December 11 – 13, 2018 | Seattle, WA
• We (Uber) are hiring! https://uber.com/careers/
Appendix
Understanding Sampling

Tracing data can exceed business traffic.

Most tracing systems sample transactions:

- **Head-based sampling**: the sampling decision is made just before the trace is started, and it is respected by all nodes in the graph

- **Tail-based sampling**: the sampling decision is made after the trace is completed / collected