分布式Tracing系统

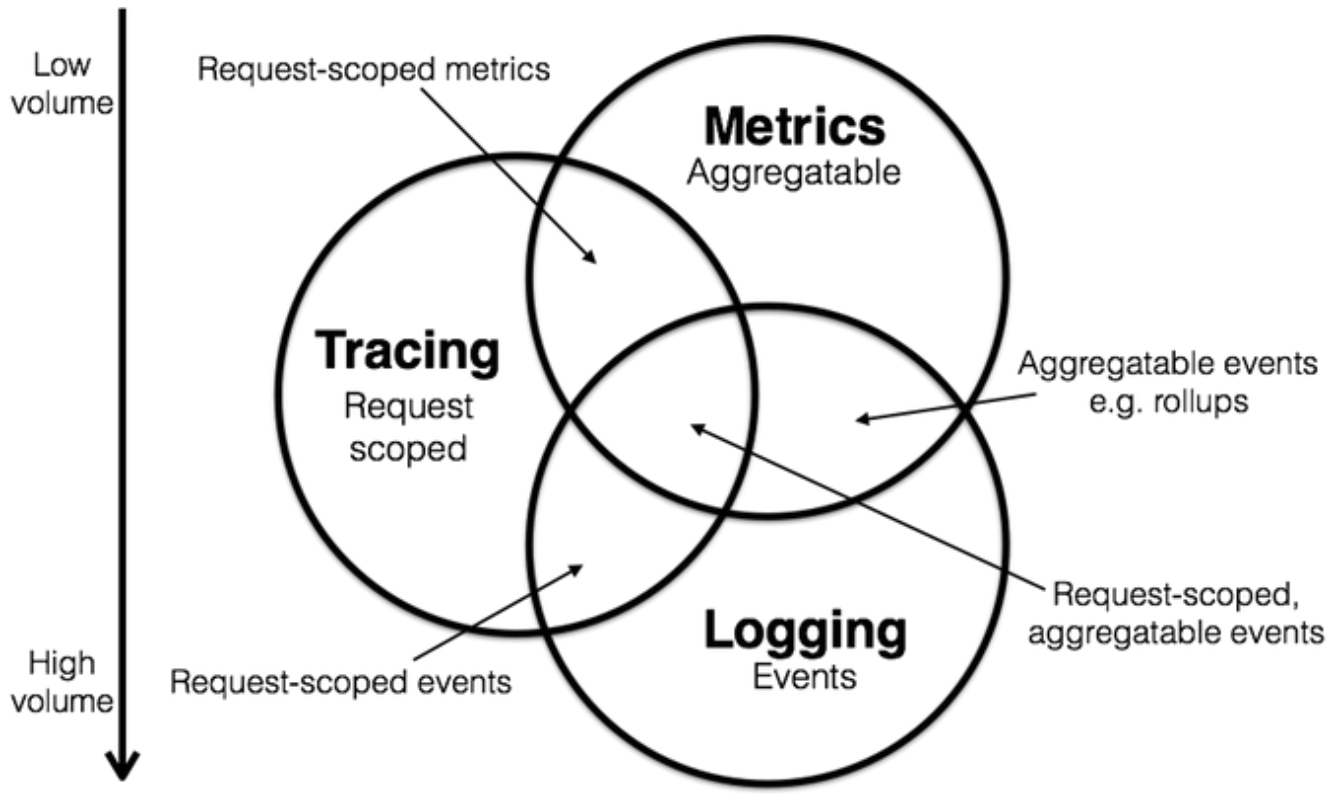
什么是Tracing

现代的互联网应用的后端架构，都向服务化转变。一方面，服务化可以降低耦合，建立故障屏障，提高开发效率，另一方面，使用大量的廉价机器已经被验证是一个经济有效的方案（在Google）。

后端服务的增多以及业务的迭代，也导致了服务之间相互依赖的复杂化。当进行故障排查，或是性能调优时，工程师需要了解**服务的调度链路**，以便于准确定位真正的问题，Tracing技术解决的就是此类问题。

Tracing，Logging，Metrics解决的问题有重合，但并不完全相同

- **Tracing：**
  - 关注的是某一个请求的链路，发生的服务请求的情况，例如一次用户登录请求，会通过API网关，发放到用户模块的登录服务，会请求数据库，缓存，或者第三方服务。Tracing关注的是这个过程在不同的服务，存储之间的调用链路，以及请求时间。
- **Logging：**
  - 关注的是具体的事件，例如请求通过API网关到达用户模块的登录服务时，登录服务会记录一次请求日志，但是日志不会考虑将所有的请求关联在一起。
- **Metrics：**
  - 度量关注的是一些可聚合的指标，例如登录服务在某一个时段内，接收的请求的QPS，以及每秒RT。



Dapper介绍

Tracing技术比较早之间就出现了，Google的一篇论文《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》，让这项技术真正开始流行，并且也明确了在服务化架构中Tracing的正确方式。

Dapper实现

用户发起的一个请求，可能经由多个服务调用，才能完成请求的处理，将结果返回给用户。

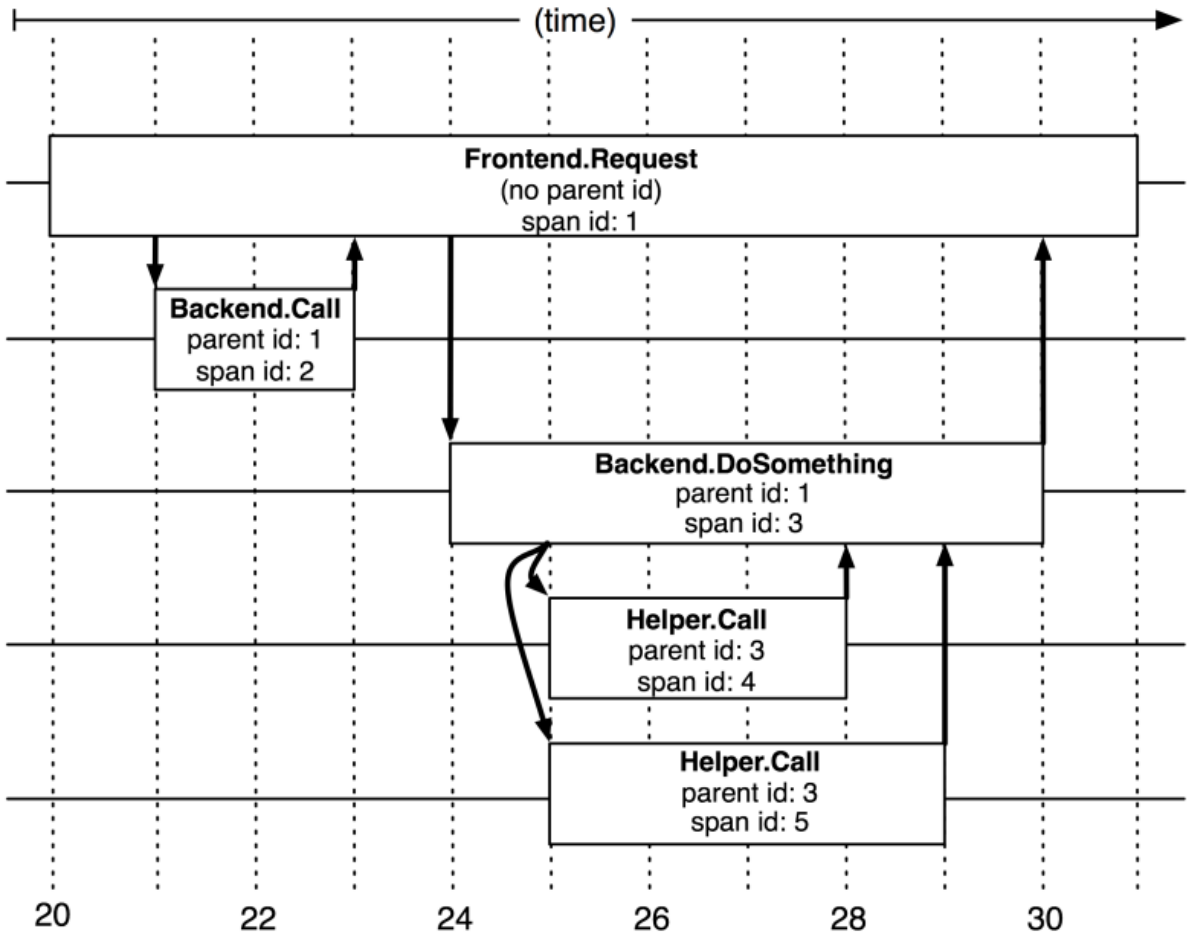


Figure 2: The causal and temporal relationships between five spans in a Dapper trace tree.

接入Dapper系统，用户请求会有一个唯一的trace_id进行标识，请求实现中的每一个服务调用，是一个span，span中会有span_id以及parent_id，服务的时间，以及其他相关数据。其中parent_id表示发起这个span的span_id, 最上层的用户请求的parent_id为0。 这些span可以组织成一个树形结构（有向无环图，DAG）。

Google的服务调用，主要由一些基础库完成，应用层使用这些基础库，调用其他的服务，因而在这些基础的RPC调用库中植入代码，即可以完成tracing数据的记录。

除了基本的调用tracing，Dapper也支持开发者自己触发，加入Annotation，方便调试。

这些tracing数据会被持久化到磁盘，服务器上的Dapper Agent后台进程会将数据发送给Dapper的服务端，并以trace id进行聚合，每一个trace id的span数据聚合成一列，存储在Bigtable中。

全量的采集，会对线上服务带了额外负载。Dapper试验表明，在高频访问的互联网服务中，低频率的采集既能覆盖几乎所有的可能性，并且对于线上的影响微乎其微。

Sampling frequency	Avg. Latency (% change)	Avg. Throughput (% change)
1/1	16.3%	−1.48%
1/2	9.40%	−0.73%
1/4	6.38%	−0.30%
1/8	4.12%	−0.23%
1/16	2.12%	−0.08%
1/1024	−0.20%	−0.06%

Table 2: The effect of different [non-adaptive] Dapper sampling frequencies on the latency and throughput of a Web search cluster. The experimental errors for these latency and throughput measurements are 2.5% and 0.15% respectively.

Dapper的应用

- 与异常系统结合：
  - Google部署了一套异常追踪系统，如果异常是发生在采样过程中，那异常追踪系统会将trace_id，span_id也同时发送给异常追踪系统，从而获取这个请求发生的上下文。
- 解决延迟的长尾效应：
  - 一次搜索请求涉及的后端服务调用时异常复杂，在Google，一次调用几千，上万的span也并不少见，类似的场景下，Dapper提供的调用链路，能够帮助工程师定位异常的服务，从而进行完成精确的修复。
- 推断服务依赖
- ...
- ...

基于Dapper实现的Jaeger(yā-gər)与Zipkin

除了Dapper之外，目前已有许多类似的Tracing系统，包括淘宝的鹰眼，Uber的Jaeger，Twitter的zipkin，AWS的X-ray等。**Jaeger**与**zipkin**是目前最为流行的两种：

	Jaeger	Zipkin
OpenTracing 兼容	兼容	兼容
Client 语言支持	官方支持PHP，Nodejs，**Python**，Go，Java，社区支持其他部分语言	官方支持，Scala，Nodejs等，社区有Python SDK
后端存储	Cassandra，ElasticSearch，Kafka（阿里云提供Collector，支持使用日志服务存储）	Cassandra，ElasticSearch，MySQL
数据传输	udp/http	http/mq
tracing实现方式	请求拦截，需要侵入代码	请求拦截，需要侵入代码

Jaeger在2017年成为CNCF（Cloud Native Computing Foundation）项目，与Kubernetes的生态环境的结合更为紧密，并且可以使用阿里云的日志服务存储数据，相对于没有运维经验的Cassandra，以及更为昂贵的ElasticSearch，这也是Jaeger的一个优势，后续搭建Tracing系统可以考虑使用Jaeger。

References

[1] Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

[2] Evolving Distributed Tracing at Uber Engineering

[3] 阿里云日志服务对接Jaeger

[4] Metrics, tracing, and logging