# Dapper，大规模分布式系统的跟踪系统

作者：Benjamin H. Sigelman, Luiz Andr´e Barroso, Mike Burrows, Pat Stephenson,

Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag

## 概述

当代的互联网的服务，通常都是用复杂的、大规模分布式集群来实现的。互联网应用构建在不同的软件模块集上，这些软件模块，有可能是由不同的团队开发、可能使用不同的编程语言来实现、有可能布在了几千台服务器，横跨多个不同的数据中心。因此，就需要一些可以帮助理解系统行为、用于分析性能问题的工具。

Dapper--Google生产环境下的分布式跟踪系统，应运而生。那么我们就来介绍一个大规模集群的跟踪系统，它是如何满足一个低损耗、应用透明的、大范围部署这三个需求的。当然Dapper设计之初，参考了一些其他分布式系统的理念，包括Magpie和X-Trace，但是我们之所以能成功应用在生产环境上，还需要一些画龙点睛之笔，采样率以及缩减通用库的代码量是我们设计dapper的关键。

自从Dapper发展成为一流的监控系统之后，给其他应用的开发者和运维团队帮了大忙，所以我们今天才发表这篇沦为，来汇报一下这两年来，Dapper是怎么构建和部署的。Dapper最初只是作为一个自给自足的监控工具起步的，但最终进化成一个监控平台，这个监控平台促生出多种多样的监控工具，有些甚至已经不是由Dapper团队开发的了。下面我们会介绍一些使用Dapper搭建的分析工具，分享一下这些工具在google内部使用的统计数据，展现一些使用场景，最后会讨论一下我们迄今为止从Dapper收获了些什么。

## 介绍

我们开发Dapper是为了收集更多的复杂分布式系统的行为信息，然后呈现给Google的开发者们。这样的分布式系统有一个特殊的好处，因为那些大规模的低端服务器，作为互联网服务的载体，是一个特殊的经济划算的平台。在这个上下文中理解分布式系统的行为，就需要对互联网进行观察，观察那些横跨了不同的应用、不同的服务器之间的彼此关联的行为。

下面举一个跟搜索相关的例子，这个例子阐述了Dapper可以应对哪些挑战。比如一个前段服务可能对上百台查询服务器发起了一个Web查询，每一个查询都有自己的Index。这个查询可能会被发送到多个的子系统，这些子系统分别用来处理广告、进行拼写检查或是查找一些像图片、视频或新闻这样的特殊结果。根据每个子系统的查询结果进行筛选，得到最终结果，最后汇总到页面上。我们把这中搜索模型称为“全局搜索”（universal search）。总的来说，这一次全局搜索有可能调用上千个服务器，设计各种服务。而且，用户对搜索的耗时是很敏感的，而查询耗时可能由任何一个子系统的低效导致。如果一个工程师只能知道这个查询耗时不正常，但是他无从知晓这个问题到底是由哪个服务调用造成的，或者为什么这个调用性能差强人意。首先，这个工程师可能无法准确的定位到这次全局搜索是调用了哪个服务，因为新的服务、乃至服务上的某个组成部分，都有可能在任何时间上过线或修改过，有可能是面向用户功能，也有可能是一些例如针对性能或安全认证方面的功能。其次，你不能苛求这个工程师对所有参与这次全局搜索的所有服务都了如指掌，每一个服务都有可能是不同的团队开发或维护的。再次，这些暴露出来的服务或服务器有可能同时还被其他客户端使用着，所以一个这个全局搜索的性能问题有可能是其他应用造成的。举个例子，一个后台服务可能要应付各种各样的请求类型，而一个使用效率很高的存储系统，比如Bigtable，有可能正被反复读写着，因为上面跑着各种各样的应用。

上面这个案例中我们可以看到，对Dapper我们只有两点要求：无所不在的部署，持续的监控。无所不在的重要性不言而喻，因为在使用跟踪系统的过程中，即便只遗漏了一小部分，那么人们对这个系统是不是值得信任都会产生巨大的质疑。另外，监控应该是7x24小时的，因为通常是哪些异常状况出现过，就很难甚至不太可能重现。那么，根据这两个明确的需求，我们可以直接推出三个具体的设计目标：

1. 低消耗：跟踪系统对在线服务的影响应该做到很小，小到可以忽略不计。在一些高度优化过的服务，即使一点点损耗也会很容易察觉到，而且有可能迫使在线服务的部署团队不得不将跟踪系统关停。
2. 应用级的透明：对于应用的程序员来说，是不需要知道有跟踪系统这回事的。如果一个 跟踪系统想生效，需要依赖应用的开发者主动配合，那么这个跟踪系统也太脆弱了，往往由于跟踪系统在应用中植入代码的bug或疏忽导致应用出问题，这样才是无法满足对跟踪系统“无所不在的部署”这个需求。面对当下想Google这样的快节奏的开发环境来说，尤其重要。
3. 延展性：Google至少在未来几年的服务和集群的规模，监控系统都应该能完全把控住。

一个额外的设计目标是为跟踪数据产生之后，进行分析的速度要快，理想情况是数据存入跟踪仓库后一分钟内就能统计出来。尽管跟踪系统对一小时前的旧数据进行统计也是相当有价值的，但如果提供新鲜的信息就可以更快对生产环境下的异常状况做出反应。

做到真正的应用级别的透明，这应该是当下面临的最挑战性的设计目标，我们把核心跟踪代码限制在很小的一部分，包括无所不在的线程、控制流和RPC库代码。使用自适应的采样率可以使跟踪系统的变得可调节，而且还可以降低性能损耗，这些内容将在第4.4节中提及。结果展示的相关系统也需要包含一些用来收集跟踪数据的代码，用来图形化的工具，以及用来分析大规模跟踪数据的库和API。虽然单独使用Dapper有时就足够让开发人员查明异常的来源，但是Dapper的初衷不是要取代所有其他的工具。我们已经发现，Dapper的数据往往侧重性能方面的调查，以致于其他工具可以针对局部起作用。

## 文献的总结

分布式系统跟踪工具的设计空间已经被一些优秀文章探索过了，其中的Pinpoint [9]、Magpie[3]和X-Trace[12]和Dapper最为相近。这些系统在其发展过程的早起倾向于写入研究报告中，即便他们还没来得及清楚地评估系统当中一些重要设计的重要性。相比之下，由于Dapper已经在大规模生产环境中摸爬滚打了多年，经过这么多生产环境的验证之后，这才是最合适的时机发表这篇论文，重点阐述亦喜爱Dapper的部署告诉了我们什么，我们的设计设计是如何实践的，以及以什么样的方式实现它才会最有用。Dappe作为一个平台，承载基于Dapper开发的性能分析工具，以及Dapper自身的监测工具，它的价值在于我们可以在回顾评估中找出一些意想不到的结果。

虽然Dapper在许多高阶的设计思想上与Pinpoint和Magpie有异曲同工之妙，但我们的实现，在这个领域中包含了许多新的贡献。例如，我们想实现低损耗的话，特别是在高度优化的而且趋于极端延迟敏感的Web服务中，采样率是很必要的。或许更令人惊讶的是，我们发现即便是1/1000的采样率，对于跟踪数据的通用使用层面上，也可以提供足够多的信息。

我们的系统的另一个重要的特征，就是我们能实现的应用级的透明。我们的组件对应用的侵入被先限制在足够低的水平上，即使想Google网页搜索这么大规模的分布式系统，也可以直接进行跟踪而无需使用额外的标注。虽然由于我们的部署系统有幸是一定程度的同质化的，所以更容易做到对应用层的透明这点，但是我们证明了这是实现这种程度的透明性的充分条件。