云计算（第三版）

CLOUD COMPUTING Third Edition

第 2 章

Google云计算原理与应用（五）

主编：刘鹏 教授

目录

2.6 大规模分布式系统的监控基础架构Dapper

- 2.6.1 基本设计目标
- 2.6.2 Dapper监控系统简介
- 2.6.3 关键性技术
- 2.6.4 常用Dapper工具
- 2.6.5 Dapper使用经验

在我们看来很简单的一次搜索实际上涉及了众多Google后台子系统，这些子系统的运行状态都需要进行监控

- **两个基本要求**

监控系统设计两个基本要求

1. 广泛可部署性（Ubiquitous Deployment）

设计出的监控系统应当能够对尽可能多的Google服务进行监控

2. 不间断的监控

Google的服务是全天候的，如果不能对Google 的后台同样进行全天候的监控很可能会错过某些无法再现的关键性故障

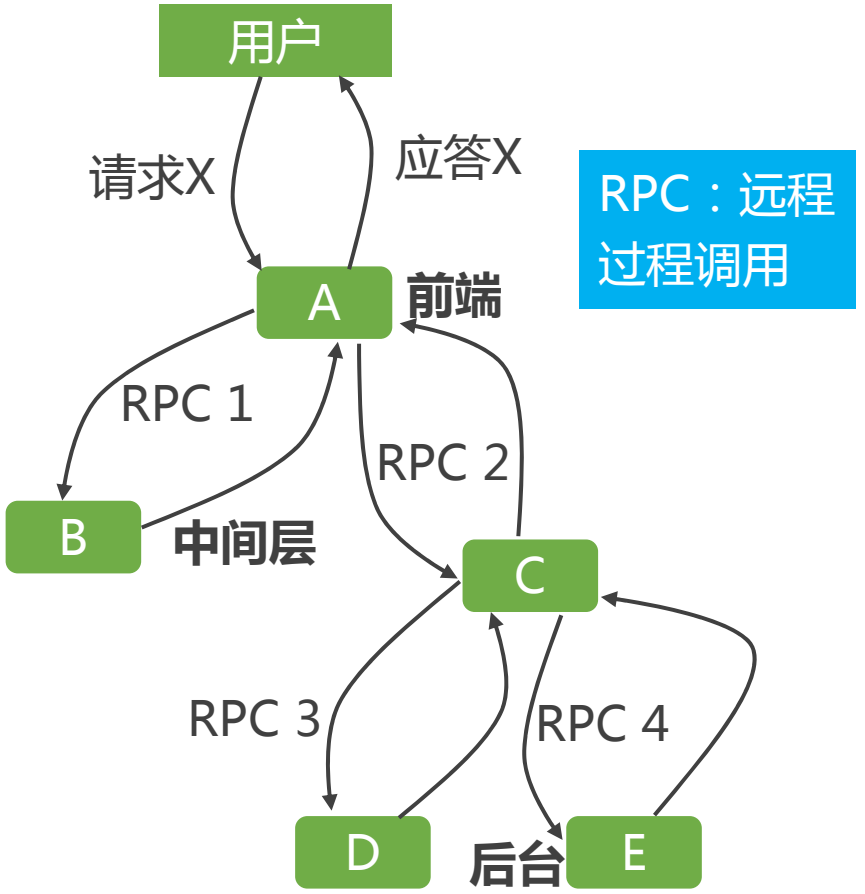
三个基本设计目标

低开销	这个是广泛可部署性的必然要求。监控系统的开销越低，对于原系统的影响就越小，系统的开发人员也就越愿意接受这个监控系统。
对应用层透明	监控系统对程序员应当是不可见的。如果监控系统的使用需要程序开发人员对其底层的一些细节进行调整才能正常工作的话，这个监控系统肯定不是一个完善的监控系统。
可扩展性	Google的服务增长速度是惊人的，设计出的系统至少在未来几年里要能够满足Google服务和集群的需求。

2.6 大规模分布式系统的监控基础架构Dapper

- 2.6.1 基本设计目标
- 2.6.2 Dapper监控系统简介
- 2.6.3 关键性技术
- 2.6.4 常用Dapper工具
- 2.6.5 Dapper使用经验

Dapper监控系统的基本概念



典型分布式系统的请求及应答过程

- 在监控系统中记录下所有这些消息不难，如何将这些消息记录同特定的请求（本例中的X）关联起来才是分布式监控系统设计中需要解决的关键性问题之一。

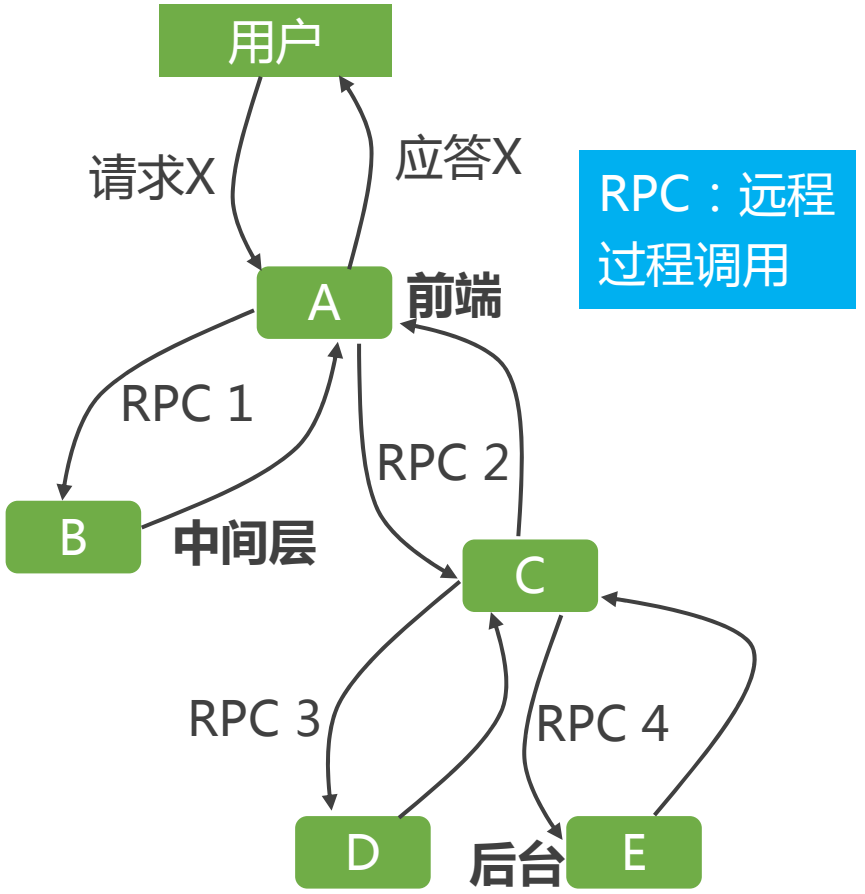
- Dapper监控系统的基本概念

黑盒

轻便，基于统计学，不准确

基于注释的监控

利用应用程序或中间件给每条记录赋予一个全局性的标示符，将相关的消息串联起来



典型分布式系统的请求及应答过程

- Dapper监控系统的三个基本概念

监控树（Trace Tree）

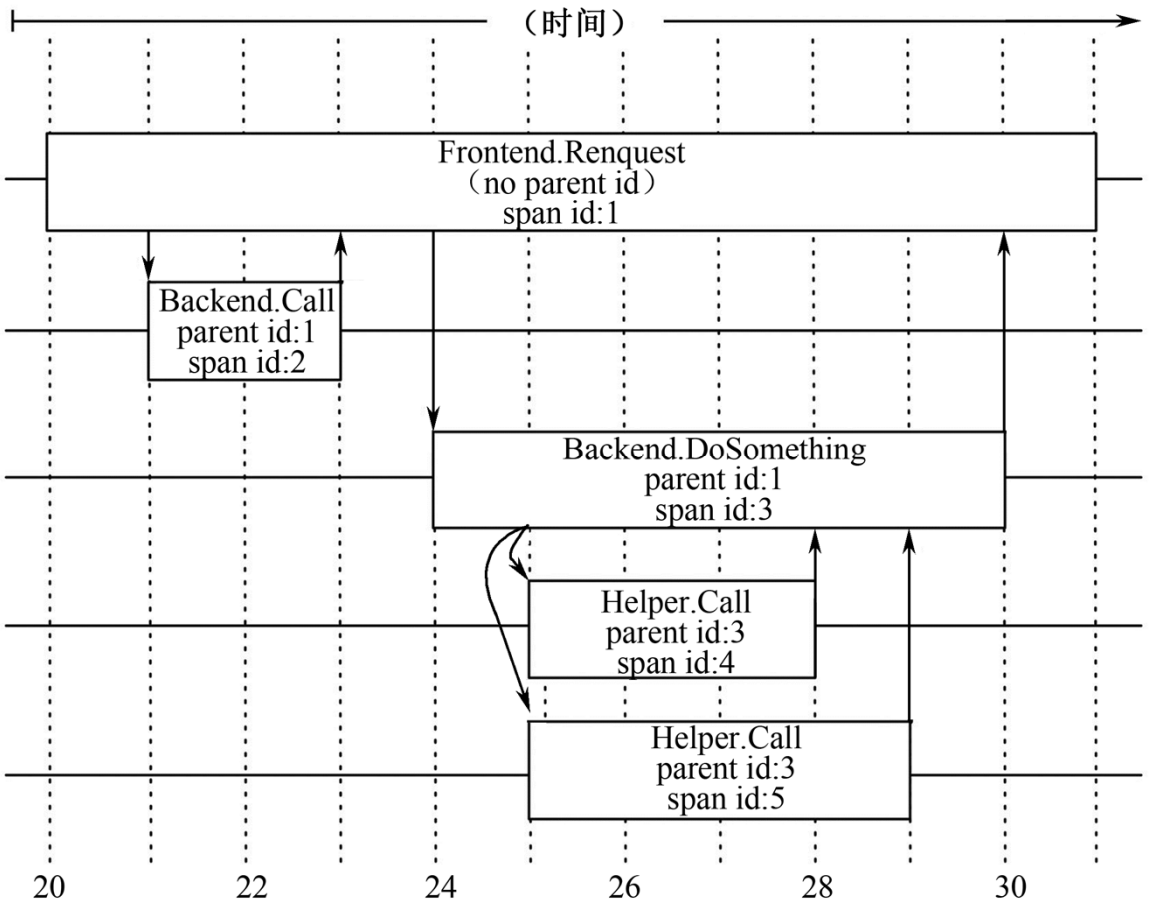
一个同特定事件相关的所有消息，按照一定的规律以树的形式组织起来

区间（Span）

树里面的节点，实际上就是一条记录，所有的记录联系在一起就构成了对某个事件的完整监控

注释（Annotation）

注释主要用来辅助推断区间关系，也可以包含一些自定义的内容



• Dapper监控系统的三个基本概念

监控树（Trace Tree）

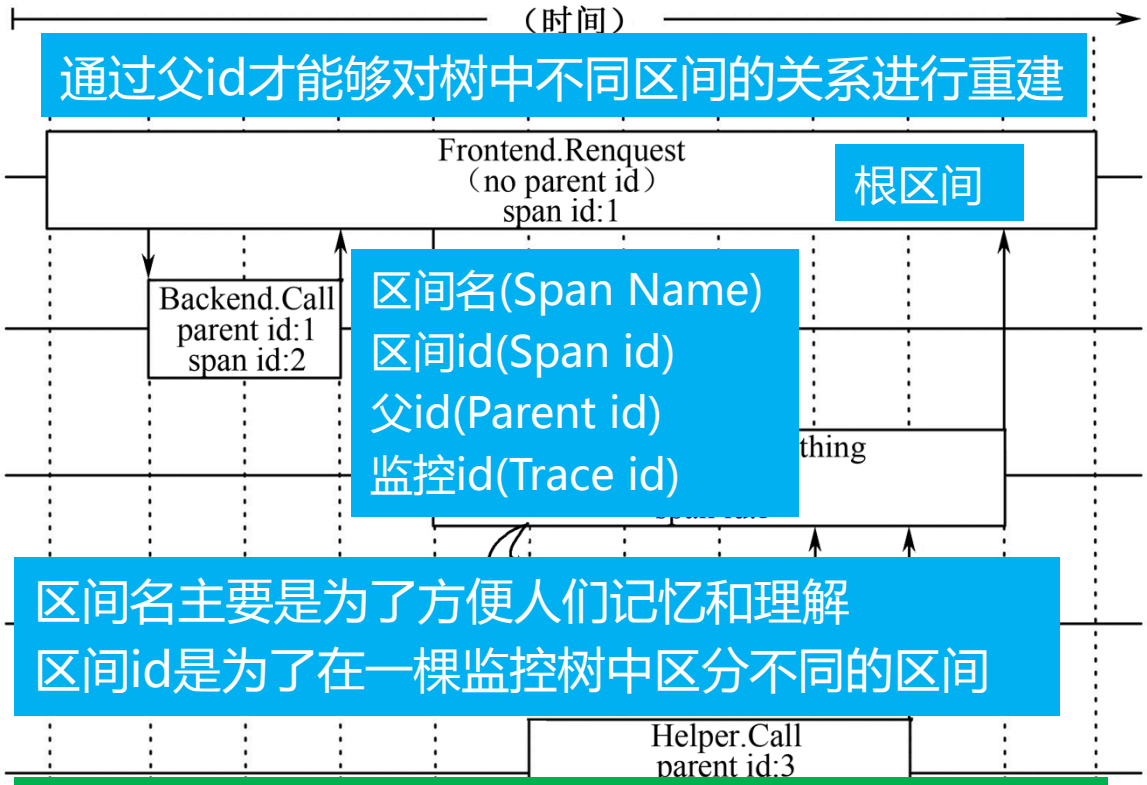
一个同特定事件相关的所有消息，按照一定的规律以树的形式组织起来

区间（Span）

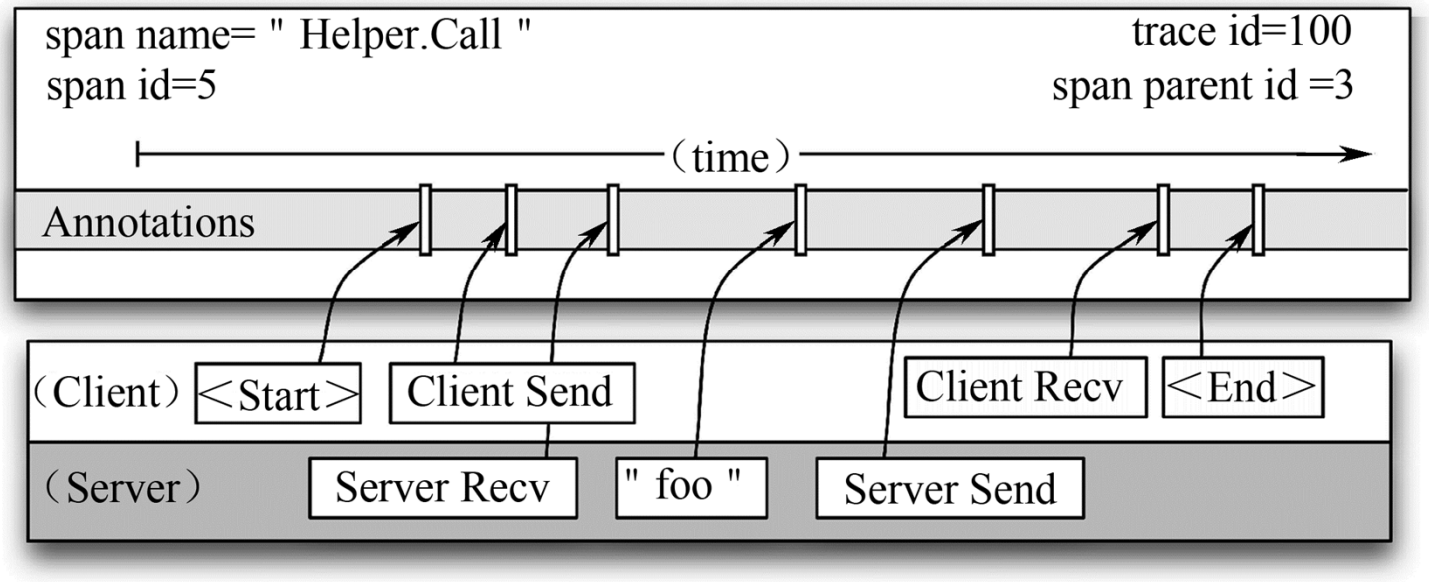
树里面的节点，实际上就是一条记录，所有的记录联系在一起就构成了对某个事件的完整监控

注释（Annotation）

注释主要用来辅助推断区间关系，也可以包含一些自定义的内容



- 区间Helper.Call的详细信息

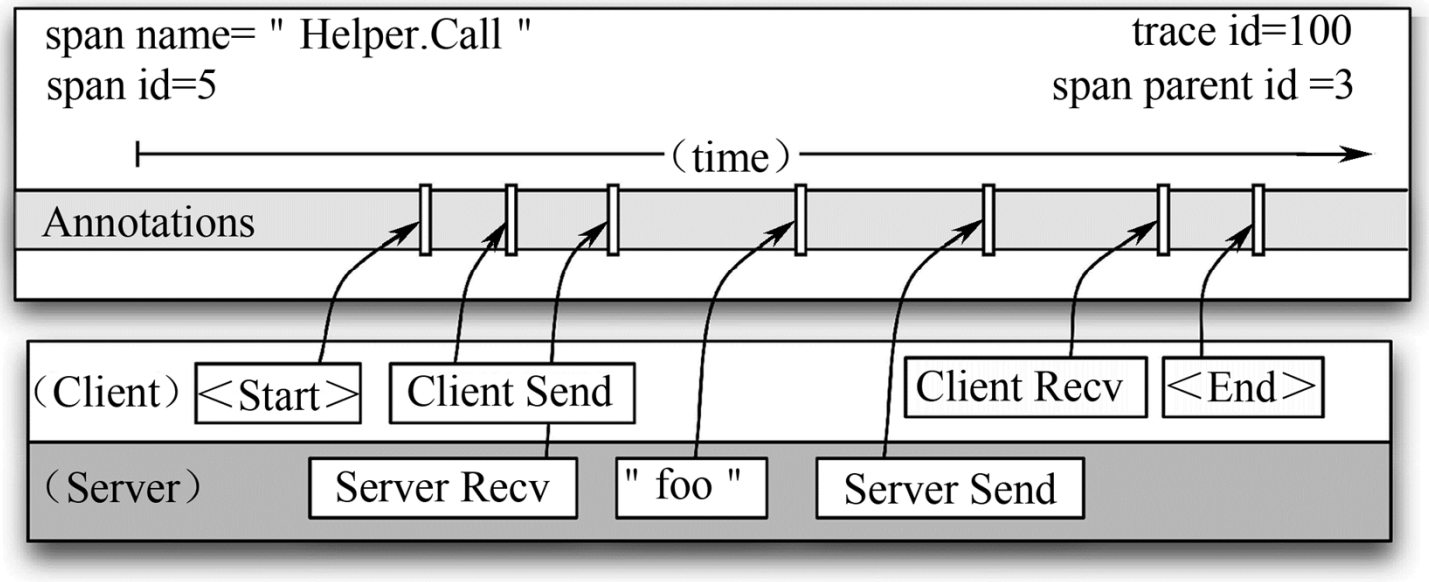


一个区间既可以只有一台主机的信息，也可以包含来源于多个主机的信息；

双主机区间：最常见，每个RPC（远程过程调用）区间都包含来自于客户端和服务器端的注释

文本方式　　键-值对

- 区间Helper.Call的详细信息



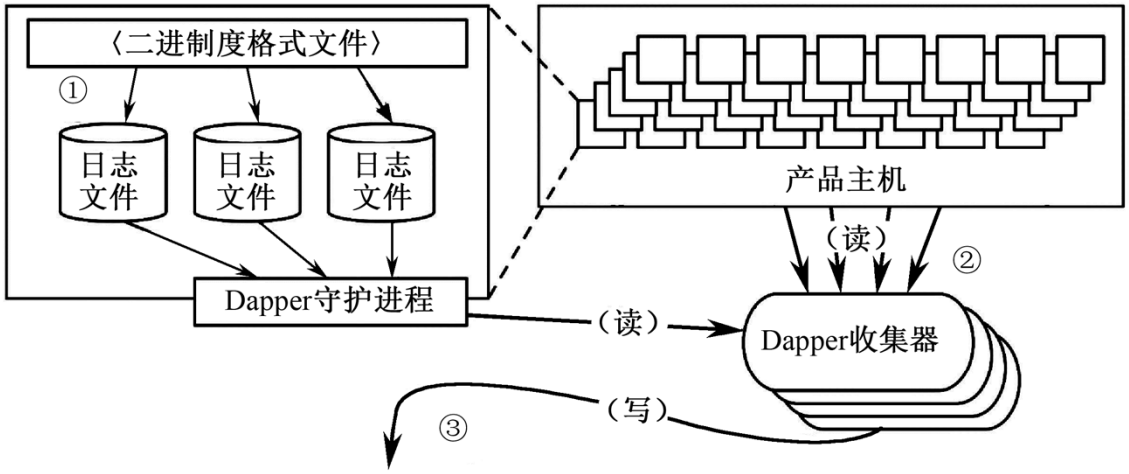
区间包含了来自客户端的注释信息：“<Start>”、“Client Send”、“Client Recv”和“<End>”，也包含了来自服务器端的注释信息：“Server Recv”、“foo”和“Server Send”

- 监控信息的汇总

（1）将区间的数据写入到本地的日志文件

（2）所有机器上的本地日志文件汇集

（3）汇集后的数据写入到Bigtable存储库中



监控id	区间12	区间23	区间34	区间45	区间56	…
123456	nil	nil	〈数据〉	〈数据〉	nil	…
246802	〈数据〉	nil	nil	nil	〈数据〉	…
357913	nil	〈数据〉	nil	nil	nil	…
…	…	…	…	…	…	…

Biotable

选择Bigtable的原因

因为区间的数目非常多，而且各个区间的长度变化很大，Bigtable对于这种很松散的表结构能够很好地进行支持。

写入数据后的Bigtable中，单独的一**行**表示一个**记录**，而一**列**则相当于一个**区间**。

监控数据的汇总是单独进行的，而不是伴随系统对用户的应答一起返回的原因：

两个原因：

1. 一个内置的汇总方案（监控数据随RPC应答头返回）会影响网络动态。一般来说，RPC应答数据规模比较的小，通常不超过10KB。而区间数据往往非常的庞大，如果将二者放在一起传输，会使这些RPC应答数据相对“矮化”进而影响后期的分析。
2. 内置的汇总方案需要保证所有的RPC都是完全嵌套的，但有许多的中间件系统在其所有的后台返回最终结果之前就对调用者返回结果，这样有些监控信息就无法被收集。

2.6 大规模分布式系统的监控基础架构Dapper

三个基本设计目标

- **轻量级核心功能库**

实现“对应用层透明”的目标

- **最关键的代码基础**是基本RPC、线程和控制流函数库的实现
- **主要功能**是实现区间创建、抽样和在本地磁盘上记录日志。
- 将复杂的功能实现限制在一个**轻量级的核心功能库**中保证了Dapper的监控过程基本对应用层透明。

- **二次抽样技术**

利用二次抽样技术可以解决**低开销及广泛可部署性**的问题。

第一次抽样

实践中，设计人员发现当抽样率低至1/1024时也能够产生足够多的有效监控数据，即在1024个请求中抽取1个进行监控也是可行的，从而可以捕获有效数据

海量数据，关注的事件出现的次数足够多

统一抽样率：不利于流量较低的服务，可能忽略关键性事件

适应性抽样率

• 二次抽样技术

利用二次抽样技术可以解决**低开销及广泛可部署性**的问题。

第一次抽样	实践中，设计人员发现当抽样率低至1/1024时也能够产生足够多的有效监控数据，即在1024个请求中抽取1个进行监控也是可行的，从而可以捕获有效数据
第二次抽样	发生在数据写入Bigtable前。 具体方法是将监控id散列（hash，压缩映射）成一个标量z（0≤z≤1），如果某个区间的z小于事先定义好的汇总抽样系数，则保留这个区间并将它写入Bigtable，否则丢弃

2.6 大规模分布式系统的监控基础架构Dapper

2.6.1 基本设计目标

2.6.2 Dapper监控系统简介

2.6.3 关键性技术

▶ 2.6.4 常用Dapper工具

2.6.5 Dapper使用经验

• Dapper存储API

Dapper的“存储API”简称为 **DAPI**，提供了对分散在区域Dapper存储库（DEPOTS）的监控记录的直接访问。一般有以下三种方式访问这些记录。

（1）通过监控id访问（Access by Trace id）

利用全局唯一的监控id直接访问所需的监控数据

（2）块访问（Bulk Access）

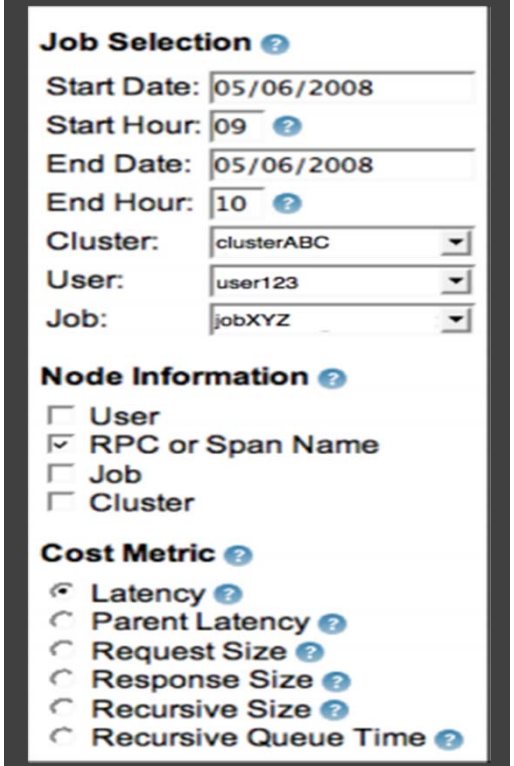
借助MapReduce对数以十亿计的Dapper监控数据的并行访问

（3）索引访问（Indexed Access）

Dapper存储库支持单索引（Single Index）

• Dapper用户界面

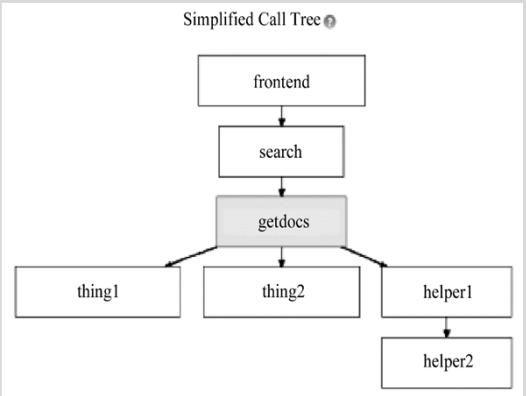
（1）选择监控对象



（2）用户对这些执行模式进行排序并选择查看更多细节

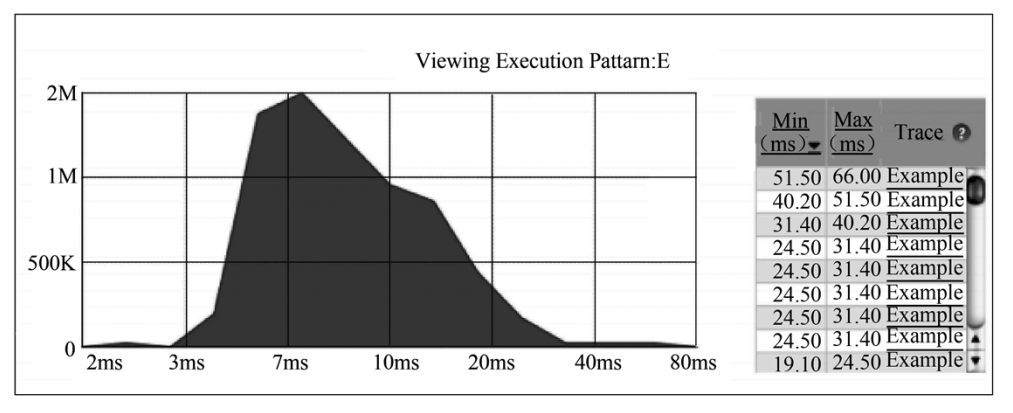
Id	Calls	Total (ms)	Global 90%ile Contribution (count)	Local 90%ile (ms)	Absolute Histogram (ms)	Scaled Histogram (ms)	View
All	40,990,720 (100.00%)	139,773,132.8 (100.00%)	4,098,118 (100.00%)	8.91			View
E	3,450,880 (8.42%)	39,437,312.0 (28.22%)	1,918,437 (46.81%)	19.17			View
R	1,658,880 (4.05%)	55,939,686.4 (40.02%)	1,658,880 (40.48%)	47.21			View

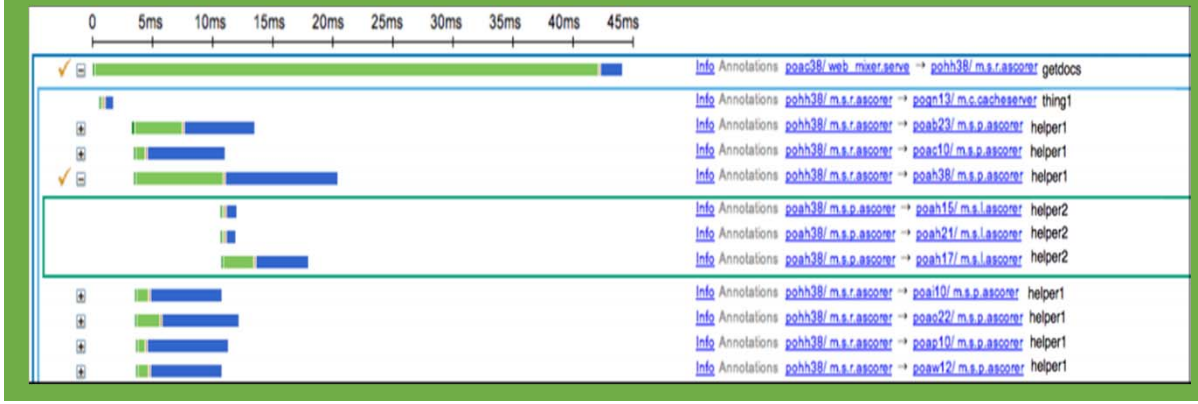
（3）分布式执行模式图形化描述呈现给用户



• Dapper用户界面

（4）根据最初选择的开销度量标准，Dapper以频度直方图的形式将步骤（3）中选中的执行模式的开销分布展示出来





（5）用户选择了某个监控样例后，就会进入所谓的监控审查视图（Trace Inspection View）

2.6 大规模分布式系统的监控基础架构Dapper

- 2.6.1 基本设计目标
- 2.6.2 Dapper监控系统简介
- 2.6.3 关键性技术
- 2.6.4 常用Dapper工具
- ▶ 2.6.5 Dapper使用经验

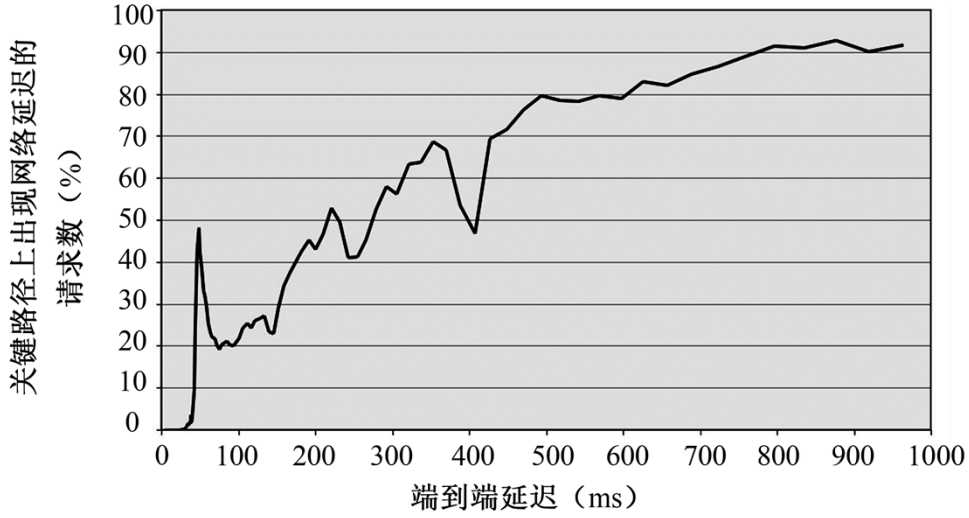
- **Dapper使用经验**

1 **新服务部署中Dapper的使用**

利用Dapper对系统延迟情况进行一系列的跟踪，进而发现存在的问题

2 **定位长尾延迟（Addressing Long Tail Latency）**

端到端性能和关键路径上的网络延迟有着极大的关系



Google最重要的产品室搜索引擎，由于规模庞大，调试非常复杂。
当用户请求的延迟过长，即延迟时间处于延迟分布的长尾时，很难判断这种端对端性能表现不好的根本原因。
——Dapper比较擅长

3 **推断服务间的依存关系（Inferring Service Dependencies）**

Google的“服务依存关系”项目使用监控注释和DPAI（存储API）的MapReduce接口实现了服务依存关系确定的自动化

4 **确定不同服务的网络使用情况**

利用Dapper平台构建了一个连续不断更新的控制台，用来显示内部集群网络通信中最活跃的应用层终端

5 **分层的共享式存储系统**

分层存储举例：Megastore→Bigtable→GFS + Chubby
导致：端用户的资源消耗模式很复杂——譬如Bigtable引起的GFS高流量可能是由一个或几个用户产生的，但是在GFS层次上，这两种使用模式是无法分开的。
没有Dapper之类的工具的情况下对于这种共享式服务资源的争用难以调试

6 **利用Dapper进行“火拼”（Firefighting with Dapper）**

“火拼”指处于危险状态的分布式系统的代表性活动。正在“火拼”中的用户需要访问最新的数据却没有时间编写新的DAPI代码或等待周期性的报告，Dapper用户可以通过和Dapper守护进程的直接通信，将所需的最新数据汇总在一起