22 | Spanner（上）：“重写”Bigtable和Megastore

2021-11-19 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩

时长 19:57 大小 18.28M

你好，我是徐文浩。

经过两个月的旅程，我们终于来到了 Spanner 面前。在这个课程的一开始，我们一起看过 GFS 这样的分布式文件存储系统，然后基于 GFS 的分布式存储，我们看到了 Bigtable 这样的分布式 KV 数据库是如何搭建的。接着在过去的三讲里，我们又看到了 Megasto 如何基于 Bigtable 搭建出来的。相信你现在也发现了，通过不断利用已经搭建好的成 统，分布式数据库的功能越来越强大，架构也越来越复杂。

不过，即使是 Megastore，它也仍然有各种各样的缺点。比如想要跨实体组实现事务 们就需要使用昂贵的两阶段事务。而由于所有的跨数据中心的数据写入，都是通过 Paxos 算法来实现的，这就使得单个实体组也只能支持每秒几次的事务。

所以，最终 Google 还是选择了另起炉灶，实现了一个全新的数据库系统 Spanner。Spanner 的论文一发表，就获得了很大的反响，成为了新一代数据库的标杆。而论文的标题也很简单明确，就叫做“Google’s Globally-Distributed Database”。

那么，在接下来的两讲里，我们就一起来学习一下 Spanner 的这篇论文的内容。**Spanner 不同于 Megastore，它是一个全新设计的新系统，而不是在 Megastore 或者 Bigtable 上修修补补。**不过，Spanner 从 Bigtable 和 Megastore 的设计和应用中，都汲取了非常多的养分，你会在它的论文里看到大量 Bigtable 和 Megastore 的影子。

好在，我们在过去的两个月里，对于 Bigtable、Paxos 算法、数据库事务，以及 Megastore 的架构都已经做了深入的学习和理解，所以 Spanner 的论文不会是那么难懂的了。

在课程中，我们会主要剖析 Spanner 这篇论文的两个主题：

> 这一讲，我们主要讲解 Spanner 的整体架构，以及这个架构是解决了 Megastore 里的哪些不足之处。这一部分主要是论文的第 1 和第 2 章节。
>
> 下一讲，我们会重点讲解 Spanner 的数据库事务，特别是跨数据中心的事务里，和“时间”这个概念有关的有趣问题。这一部分主要是论文的第 3 和第 4 章节。

在学完这一讲之后，希望你能够和 Megastore 的设计作一个对照，对一个需要在全球部署的分布式数据库，需要面临的问题和挑战有一个深入的认识和理解。虽然我们可能都不一定有机会去参与开发这样一个数据库，但是理解这个系统是如何设计的，对于我们未来参与去设计新一代的数据基础设施会有莫大的帮助。

我们可以部署一个全球的 Megastore 吗？

我们先来看看 Spanner 到底是一个什么样的宏伟的系统。在论文的引言（Introduction）部分，是这么说的：“Spanner 是一个 Google 设计、开发、部署的可伸缩的、全球分布式数据库……Spanner 的可伸缩性是为百万级别的服务器、上百个数据中心、万亿级别的数据量进行打造的”。

说实话，当看到论文的这第一段话，我就有些被镇住了。当时，我们公司在国内也在多个城市有 3 个数据中心，一个 Hadoop 集群也有上千台服务器。不过，和 Google 列出的这

些数字一比，实在是让人相形见绌。

正好，我们刚刚学过 Megastore 的论文，那么我们可不可以直接把单个的 Megastore，部署成一个“全球的分布式数据”，做到伸缩到“百万级别的服务器、上百个数据中心、万亿级别的数据量”呢？

我想了一下，觉得应该做不到，**核心的挑战，还在于网络延时**。在 Megastore 里，每个数据中心，都是一个 Paxos 集群里的一个副本，而且是一个完全副本。这样，在每一次的数据写入时，我们都需要有网络请求到所有的数据中心。

如果这个距离仅仅是上一讲的上海到广州可能还好，一个往返可能也就是 30~50 毫秒。但是如果每次写入，都需要从上海向欧洲、美洲发起网络请求。那么，我们一个数据库事务的写入，就要 100~200 毫秒，甚至更长的时间，而这就会进一步拖累 Megastore 原本就不算出色的单个实体组下的每秒事务写入量了。

所以，**为了容错，我们需要让数据在多个数据中心进行同步复制，但是为了避免过大的网络延时，我们又不能把数据放在太多的数据中心里。**所以，在整个数据库层面，我们要对数据进行分区，每个分区会有跨数据中心的多个副本。但是我们并不会把数据，在所有的数据中心都进行复制。

那么最佳的策略，就是**让数据副本，尽量放在离会读写它的用户近的数据中心**就好。而这个思路，就是 Spanner 的整体架构设计思路。

Spanner 的整体架构设计

部署好的一个 Spanner 数据库，在论文里被称之为是一个“宇宙”（Universe）。在这个“宇宙”里，会有很多很多个区域（Zone）。每一个 Zone 里，都有一套类似于 Bigtable 这样的分布式数据库的系统，你可以把它看成是在一个内网内的一套类 Bigtable 的部署。而很多套这样的类 Bigtable 的部署，就共同组成了一个 Spanner 宇宙。

那么，Spanner 要复制数据，就是把数据复制到不同的“Zone”里面去。一个数据中心里，可以有多个“Zone”，也就是多套不同的类 Bigtable 的部署。注意，尽管多个“Zone”可能是在一个数据中心里的，但是它们互相之间是物理隔离的。如果要打一个比方的话，你可以把它们看成是部署在同一个数据中心的两栋楼。不同 Zone 里面的两台服务器，并不能直接互相访问。

而一个 Zone 里的服务器，主要分成了三种类型，分别是：

Zonemaster，负责把数据分配给 Spanserver；

Spanserver，负责把数据提供给客户端；

Location Proxy，用来让客户端定位哪一个 Spanserver 可以提供自己想要的数据。

其实这个组合和我们之前学习的 Bigtable 非常类似。Zonemaster 就好像 Bigtable 里的 Master 节点，负责分配 Tablet；Spanserver 就好像 Tablet Server，负责提供数据和在线服务。而 Location Proxy 则是新出现的节点，它承担了原先 Chubby 和 METADATA 表的功能。

只不过，相比原先的 METADATA 表是直接作为 Bigtable 里的一张表出现，Location Proxy 则是把这个功能完全独立了出来。

Bigtable	Spanner	作用
Master	Zonemaster	分配数据给不同Tablet Server/Spanserver
Tablet Server	Spanserver	管理Tablet，提供在线服务
Chubby+METADATA表	Location Proxy	告诉我们数据在哪个Tablet里

极客时间

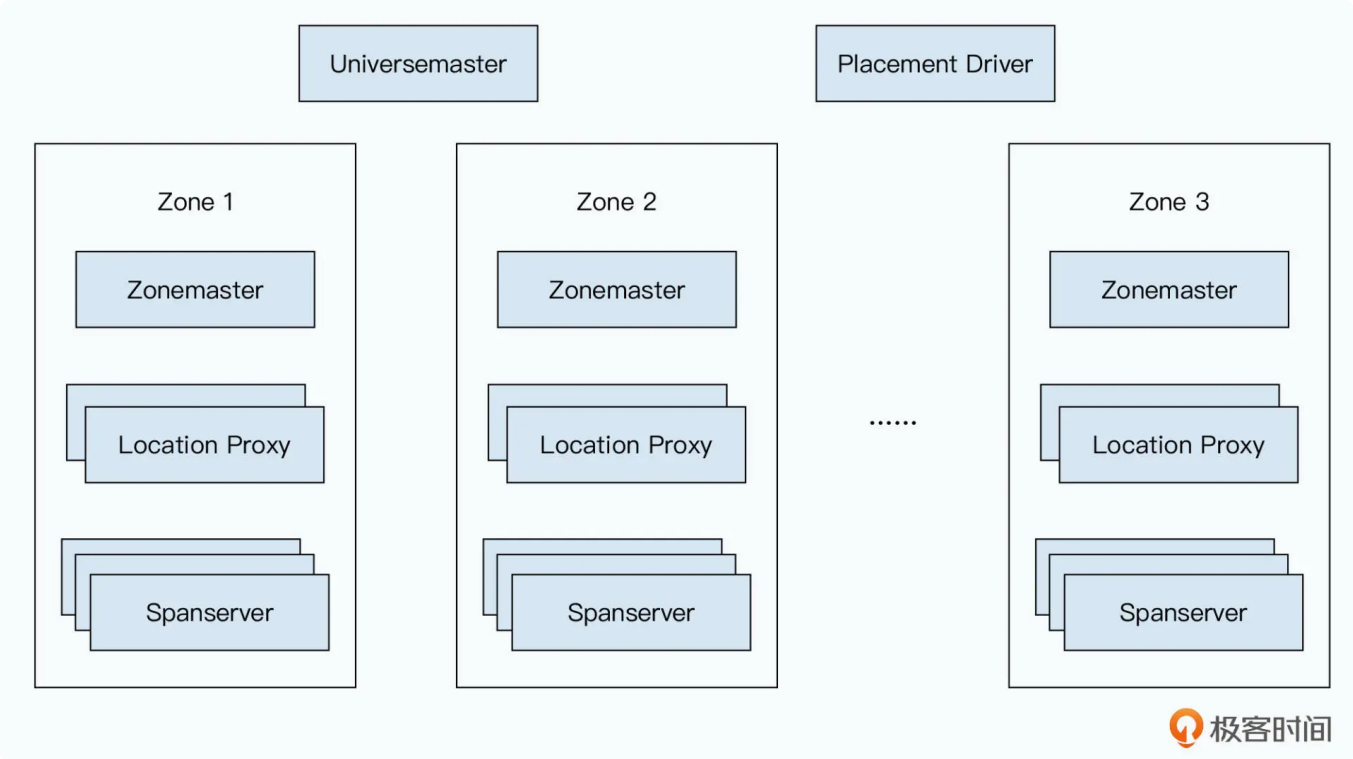
其实分布式数据库的基本架构都是相似的，通过Master管理数据分区，通过Tablet Server提供分区数据的在线服务，然后数据在哪一个分区都不从Master查询而是独立剥离出来，避免Master成为系统性能瓶颈

一个 Zone 里，只会有一个 Zonemaster，但是会有多个 Location Proxy，但是又有成百上千个 Spanserver。而每个 Spanserver 里，又和 Bigtable 里的 Tablet Server 一样，会负责管理 100~1000 个 Tablet。

而在很多个 Zone 之上，整个 Spanner 宇宙里，还会有两个服务器。

一个叫做 **Universemaster**，也就是这个宇宙的 Master，它只是提供一系列的控制台信息，主要是 Zone 的各种状态信息。这些信息就像我们在 MapReduce 的论文里见过的 JobTracker那样，通过暴露各类信息，方便我们去监控和调试整个“宇宙”。

还有一个服务器，叫做 **Placement Driver**，它的作用就是调度数据，也就是把数据分配到不同的 Zone 里。这个分配策略，是使用数据库的应用程序就可以定义的。



论文中的图1，也就是Spanner的Server的组织结构图

另外在 Spanner 的论文里，列出了它支持的一些控制的策略，包括：

明确指定哪些数据中心应该包括哪些数据。这个设置，是把数据在不同 Zone 内进行分布的权利，放到了应用开发人员的手里。

申明要求数据距离用户有多远（为了控制用户读取数据的延时）。这个相当于动态根据用户的访问延时，去调度对应数据应该放在哪些 Zone 里。

申明不同数据副本之间距离有多远（为了控制用户写入的延时）。这个就是我们前面讲过的，我们不应该在上海的用户写入数据的时候，一定要把数据复制到欧洲一份，这样会大大影响数据库事务写入的效率。

申明需要维护多少个副本（为了控制系统的可用性和读操作性能）。因为副本越多，可用性越高，即使因为自然灾害损失了多个 Zone，系统也是可用的。而且副本越多，意味着世界各地的用户，都能低延时地读到数据。

其实，在这些策略里，第一个策略在 Megastore 上也能实现，我们只要自己在 Megastore 上去实现“分库分表”，把数据拆分到多个 Megastore 里就好了。

但是，如果要真正做到一个“易用”的数据库，我们就不应该让应用开发人员，去关心底层数据到底存放在哪里。而应该像 Spanner 的后几种策略一样，让开发人员去申明期望的读延时和写延时是多少，然后让 Spanner 自己去调度数据，来满足这个需要。

Spanserver 的实现

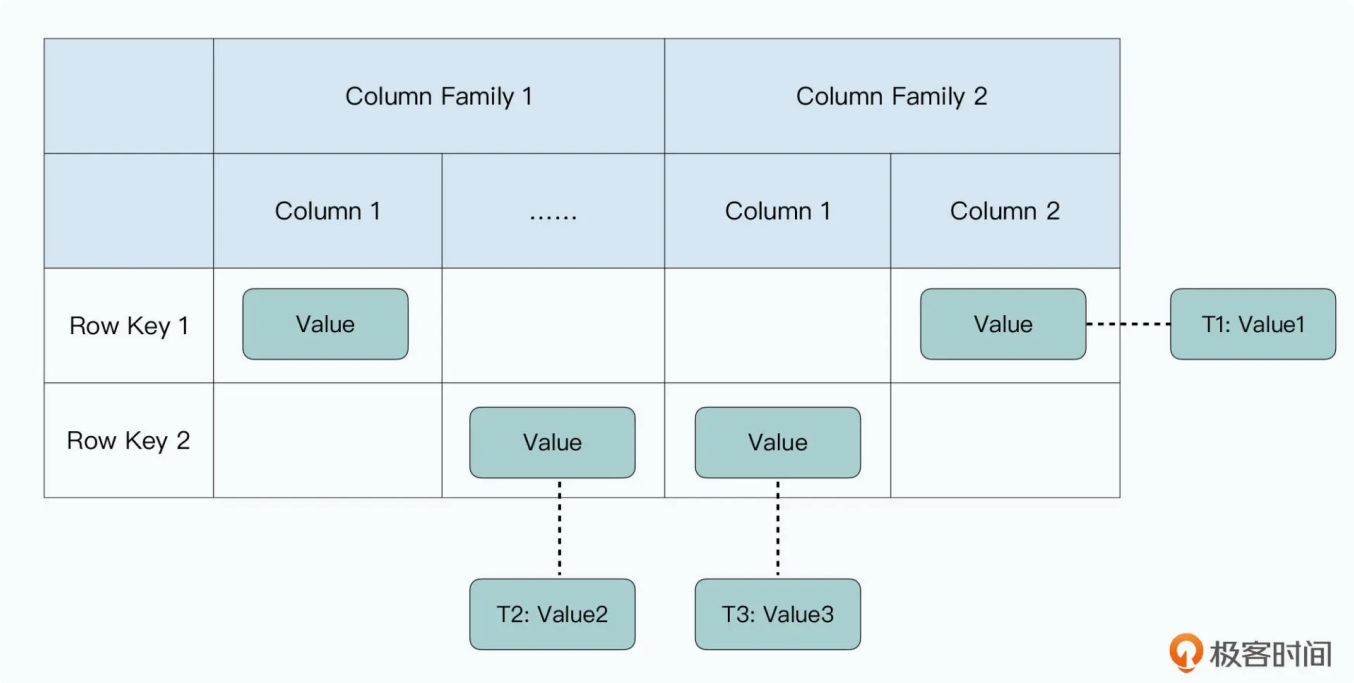
了解了 Spanner 的整体架构，我们再来深入看看 Spanserver 的实现。前面我们讲过，一个 Zone 里其实就是一个类似于 Bigtable 的部署，Spanserver 就类似 Bigtable 里的 Tablet Server。不过，Spanner 里的 Zone 和一个 Bigtable 还是有一些差别的。

Spanserver 的架构

首先是**数据模型**。Spanner 的底层数据存储，是一个 B 树这样的数据结构，以及对应的预写日志（WAL）。Spanner 没有像 Bigtable 那样，采用 SSTable 这样的 LSM 树。所以 Spanner 的数据模型，更像是一个**关系型数据库**，而不是一个 KV 数据库。

在 Bigtable 里，我们看到每一个列的值，都有一个时间戳，在 Megastore 里我们也利用这个时间戳，来实现数据的多版本和 MVCC 下的隔离性。但是，作为一个数据库，其实我们的**“版本”应该是在整条数据上**，而不是某一列的数据上，我们的**数据库事务，针对的也是整条数据记录**。

否则，我们在实践中就会遇到一些繁琐的操作，就是在更新数据的时候，需要先读出所有列的数据，再重新用一个新版本写回去。



Bigtable里，不同的Column的Value里的TimeStamp可能并不相同
那么“最新”版本的数据，到底是只包含T3这个时间戳？还是包含所有最后一个时间戳的Column的值呢？

所以，在 Spanner 的实现里，对应的时间戳也就是在整条数据上，也就是论文里列出的：

```
1 (key:string, timestamp:int64) ⇒ string
```

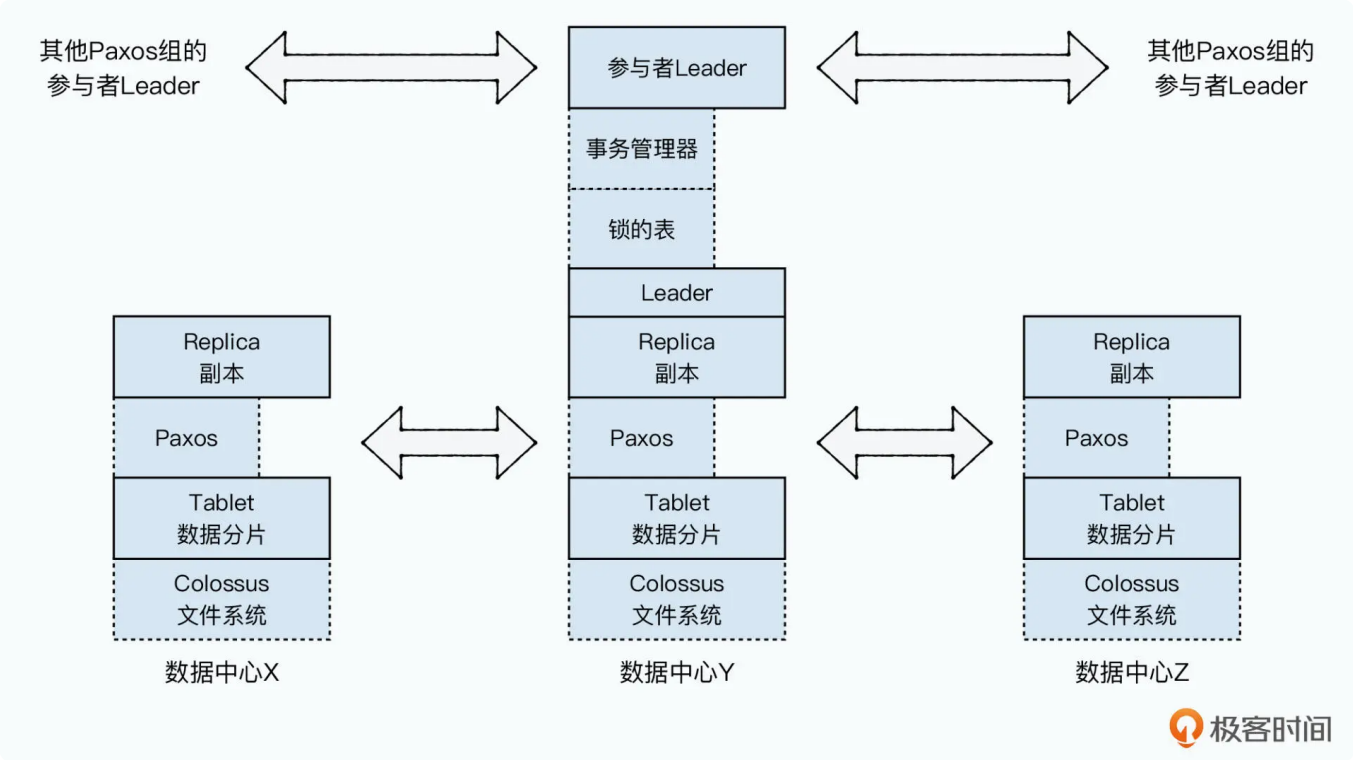
而不像是 Bigtable 那样，会把列族（Column Family）和 Column（列）也放到映射关系的时间戳之前。

而 Spanner 的底层数据存储，则是最终落地在 GFS 的后继者，一个叫做 Colossus 的分布式文件系统里。不过可惜的是，对于 Colossus，Google 并没有公开对应的论文，我们找不到太多的资料。

然后，为了保障数据的同步复制，Spanserver 自然也是采用了 **Paxos 算法**。这个 Paxos 算法，是作用在每一个 Tablet 上的。也就是在这个场景下，每一个 Tablet 相当于是 Megastore 里面的一个实体组。不过，它的 Tablet 里就不像实体组那样，只能有一条根实体的记录了。这个，会对 Spanner 的事务功能产生很大的影响，不过关于事务的部分，我们就要放到下一讲来讲了。

Spanner 的 Paxos 算法实现，也是采用基于 Leader 的算法。不过，这个 Leader，并不是在每一轮事务的时候指定的。因为 Spanner 和 Megastore 不同，不是一个实体组就是一个迷你数据库。一个 Paxos 状态机，需要管理整个 Tablet 里的很多条记录。所以，**这个 Leader 是类似于 Chubby 锁里面的“租约”的机制**。

一个租约是在 0~10s 之间，这段时间之内，Leader 并不会改变。我们的数据写入，都是从 Leader 发起的，但是所有的其他副本，也都会拥有完整的数据，所以 Spanserver 读数据，可以从任意一个副本进行。



论文里的图2，Spanner的数据复制的架构，其中的数据库事务部分，我们会放到后面来单独讲解

论文里写了，在数据写入的时候，Spanner 其实会写入两份日志，一份是 Paxos 日志，一份是 Tablet 日志，当然这个会在后续改进。不过，这个其实凸显出了一点，那就是 Spanner 不像 Megastore 那样，是基于 Bigtable 这个间接层来实现事务日志和数据存储的，而是**完全重新设计开发了一个支持同步复制的底层存储引擎**。

Spanner 的数据调度

一个 Tablet 和它的所有副本，在 Spanner 里组成了一个 **Paxos 组**（Paxos Group）。Spanner 的一个重要的设计，是可以实现“数据调度”，也就是把数据记录，调度到不同的数据中心里。而这个调度，就是在不同的 Paxos 组之间进行的。

当然，不同的 Paxos 组，可能会涵盖相同的 Zone。所以，**本质上，Spanner 的数据是在各个 Zone 之间混合存储的**，而不是简单地进行像 MySQL 集群那样“分库分表”的存储方式。

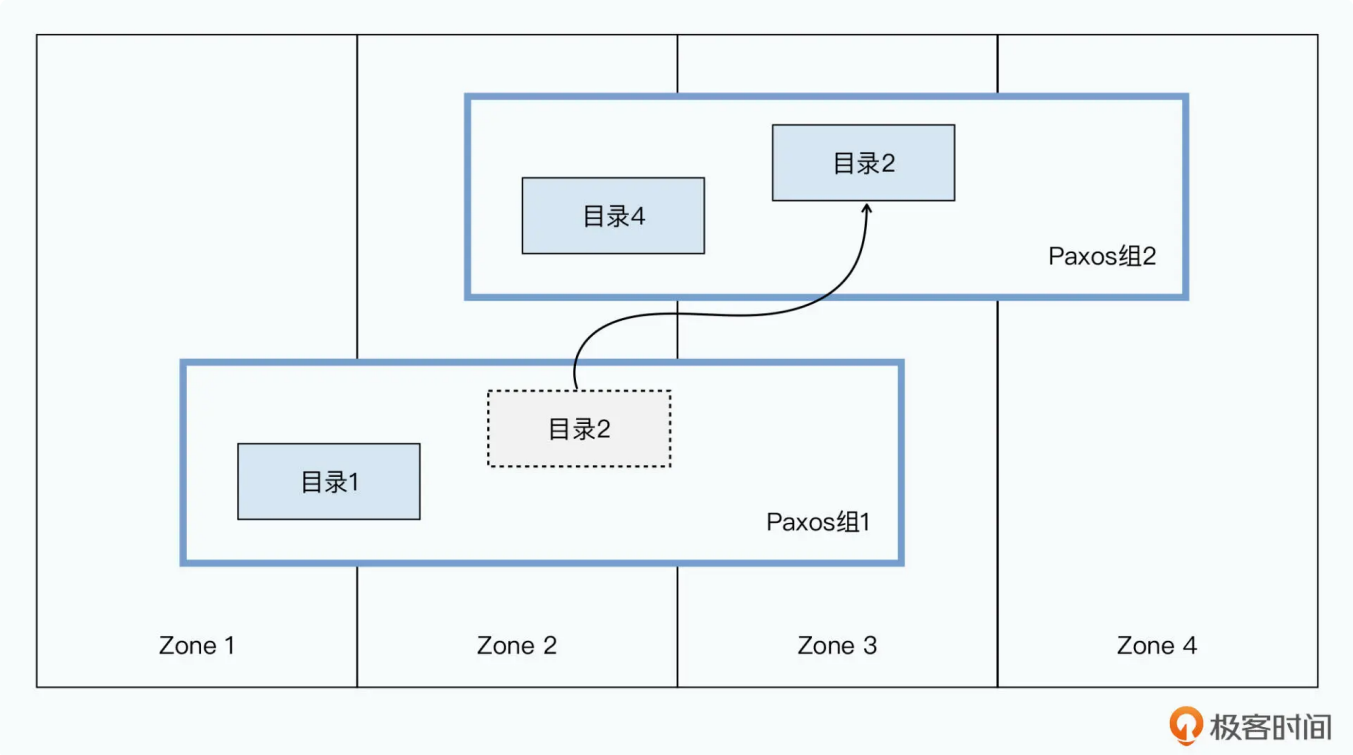
不过，Spanner 的数据调度，并不是一条条记录去调度的，而是一小片一小片连续的数据去进行调度的。一小片连续的行键，在 Spanner 中被叫做是一个“**目录**”（Directory）。我们去调度数据的时候，是一个一个目录地调度。而一个 Paxos 组里，会包含多个目录。

因为目录可以被调度走，所以一个 Spanner 的 Tablet 和 Bigtable 的 Tablet 并不一样。Bigtable 的一个 Tablet 里，所有的行键是连续的。但是在 Spanner 里，并没有这个要求，只有在里面的某一个目录里，行键是连续的。而不同的目录之间，可以是离散的。

之所以这样做，是因为虽然通常数据的访问有局部性，但是这个局部性不一定是空间局部性。也就是不一定所有连续的行键数据，都会频繁地被一起访问。

而在 Spanner 这个设计下，我们可以把那些频繁共同访问的目录，调度到相同的 Paxos 组里。这样无论是读数据，还是写数据，都可以在一个 Paxos 组里完成，进而可以提升读写性能。**我们可以根据实际数据被访问后的统计数据，来做动态的调度，而不是局限于在设计数据表结构的时候，必须要万无一失。**

而整个数据在不同 Paxos 组之间的转移，则是通过一个 **movedir 的后台任务**。这个任务也不是一个事务，这是为了避免在转移数据的过程里，阻塞了正常的数据库事务读写。它的策略是先在后台转移数据，而当所有数据快要转移完的时候，再启动一个事务转移最后剩下的数据，来减少可能的阻塞时间。



论文中的图3，数据在不同Paxos组之间的调度和迁移，是以“目录”作为单位的

不过，虽然 Spanner 定义目录是可以指定我们前面所说的，控制副本放在哪些 Zone 里的最小单元，但是实际上，当一个目录变得太大的时候，Spanner 还会再进行**分片存储**。不同分片还会存储到不同的 Paxos 组里，也就是不同的 Zone 和服务器里。我们的 movedir

进程转移的也是分片而不是整个目录，所以在具体的实现上，其实 Spanner 会比这个抽象的模型更加复杂。

Spanner 的数据模型

了解了 Spanner 的 Spanserver 的架构，以及对应的数据转移策略，我们最后再来看看 Spanner 的数据模型。其实 Spanner 的数据模型和 Megastore 的基本一致，因为 Spanner 设计的时候，Megastore 的数据模型已经在 Google 内部得到普及了。

你通过下面的 Schema 就可以看到，和 Megastore 里一样，在论文里 Google 还是给出了一个相册的例子：

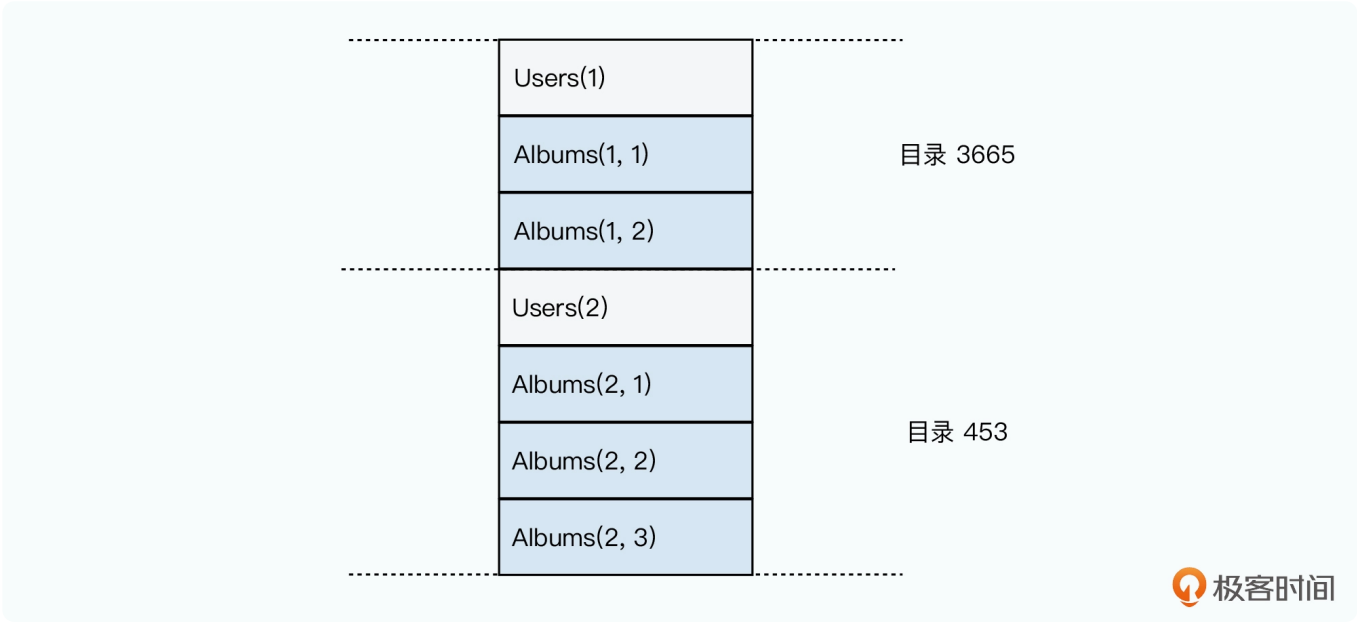
> User 表，通过一个 uid 作为主键，也就是数据表的行键；
>
> Albums 表，也就代表一个相册，使用 uid 和 aid 作为联合主键，并且通过 INTERLEAVE IN PARENT Users ON DELETE CASCADE 指明了它和 Users 表的关系。

在这其中，INTERLEAVE IN 是告诉数据库，父表的行键和子表所有相同行键的数据，共同构成了一个目录。而 ON DELETE CASCADE 则更进一步，不仅申明了关系，而且让 Spanner 可以去做级联删除，一旦父表的 User 的某一行数据删除了，对应的子表里的记录也就删除了。

复制代码

```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT65 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```



来自论文中的图4
其实和Megastore里面我们看到的Schema定义很像，一对多的外键关联，是明确申明定义的
这样能够方便Spanner管理数据的布局，以及目录数据的调度

而这个关系的声明，就使得 Spanner 可以直接根据这个信息，去对应地调度数据，从而让相同的父子行的数据，都会在一个目录里，来保障实际的访问性能。

小结

好了，相信到这里，你对 Spanner 的整体设计已经有了一个清楚的理解。在这一讲的一开始，我们就看到了 Megastore 其实很难做到 Spanner 的设计目标，也就是一个“全球”的、上百数据中心、万亿级别的数据库。

Spanner 的整体架构，其实是把多个 Zone 里的一个个分布式数据库组合在一起。相比于 Megastore，每一个数据中心都拥有完全相同的副本，Spanner 的每个 Zone 里的数据，则可能是完全不同的。Spanner 会把不同的数据，根据应用设置的延时策略，调度到不同的数据中心。对于应用开发人员来说，只需要明确自己的应用需求，而不需要去关心底层数据究竟是如何分布的。

在单个 Zone 内，Spanner 拥有 Zonemaster、Spanserver 和 Location Proxy 三种不同类型的服务器，它们可以对应到 Bigtable 里的 Master、Tablet Server 以及 METADATA 数据表。**而多个 Zone 之上**，Spanner 会通过 Universemaster 来作为控制台，并且提供 Zone 的各种信息方便 debug，以及通过 Placement Driver，进行实际的数据分布和调度。

在 Spanserver 的实现里，Spanner 没有采用 Bigtable 这样稀疏列 +LSM 树的组合，而是采用了传统关系型数据库会使用的 B 树，以及单行数据整个作为一个值的模型。

在数据的同步复制层面，Spanner 是把一个 Tablet 作为一个单元，而不是像 Megastore 那样以一个实体组作为单元。在算法上，Spanner 自然也还是使用 Paxos，但是相比于一个实体组一份 Paxos 日志，Spanner 的 Paxos，是在 Tablet 这个单位上的。所以在 Paxos 算法上，Spanner 也没有使用在上一次数据写入的时候，写入一个 Leader 的方式，而是选择为 Leader 设定了一个**租期**的策略。

在数据调度层面，Spanner 选择了把数据分成一小片一小片的目录，在不同的 Paxos 组里调度目录。也就是说，它既不是调度一个更大的 Tablet，也不是调度某一条记录，而是通过目录这个介于两者之间的单位。而这个策略，也使得 Spanner 的单个 Tablet 里的数据，不一定是连续的行键。

而在数据模型层面，Spanner 则基本继承了 Megastore 的选择。一方面，是因为 Megastore 已经在 Google 内部普及，选择向前兼容是一个对开发人员最友好的方式。另一方面，不同数据表之间明确对应的父子关系，有助于 Spanner 去调度数据，把经常要共同访问的数据，放在相同的地区或者说 Zone 里面。

我们可以看到，Spanner 的设计和实现，都有着 Bigtable 和 Megastore 的影子。但有针对自己的设计目标，Spanner 实际上重新做了设计。

相较于 Megastore 是在 Bigtable 上进行“缝缝补补”，Spanner 作为一个全新设计的系统，没有背上过多沉重的包袱。相信这一讲对你来说，也并不难理解，特别是对照前面的 Bigtable、Chubby 和 Megastore，你会发现大部分知识点都是重合的。**甚至我们可以说，其实 Spanner 就是重新梳理了 Google 内部的数据库需求，把 Bigtable 和 Megastore 重写了一遍，是一个 Bigtable+Megastore 的 2.0 版本。**

不过，整体架构上的设计，还不是 Spanner 整个系统中最重要的一点。Spanner 对于数据库事务，特别是跨地域，有大量网络延时的数据库事务的解决方案，才是它最核心的创新点。这个，就请一起和我在下一讲里学习吧。

推荐阅读

这一讲的推荐阅读，我还是会推荐你回到Spanner 论文的原文，特别是对照着我们刚刚学习过的 Megastore，先仔细阅读论文的第一部分 Introduction 和第二部分 Implementation。相信经过过去两个月的学习，对你来讲，原本看起来复杂的 Spanner 已经是小菜一碟了。对 Spanner 整体的架构理解，也能帮助你串联过去已经学习过的 Bigtable、Chubby，以及 Megastore 了。

思考题

学完了 Spanner 的设计之后，你能想一下，通过完全重新设计底层的 Spanserver，而不是复用 Bigtable 作为事务日志和 KV 存储层，Spanner 的数据写入性能，能够在哪些方面获得显著的提升呢？

欢迎在留言区分享出你的思考和答案，也欢迎你把今天学到的内容和你的老师、朋友分享、讨论，共同学习成长，一起进步。

上一篇 21 | Megastore（三）：让Paxos跨越“国界”

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。