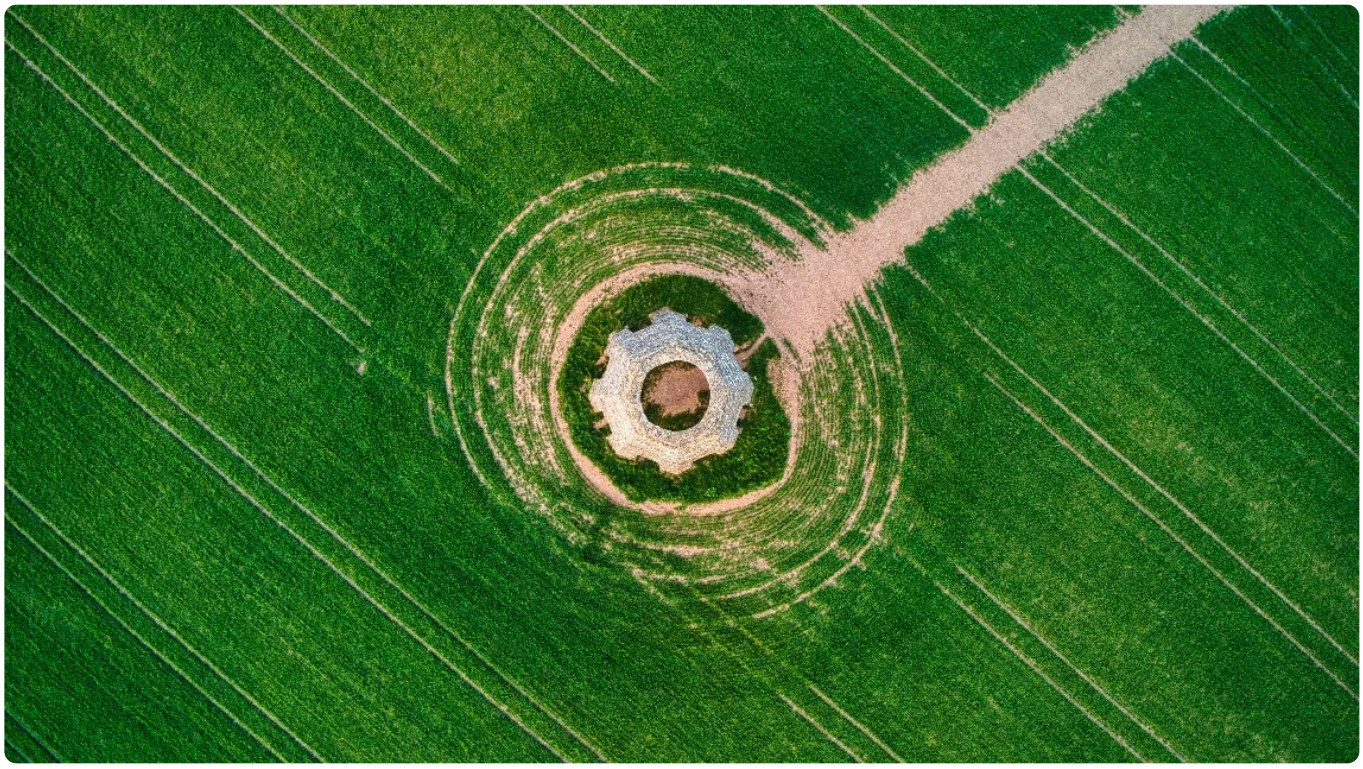
复习课（十）| Spanner

2021-12-29 黄金

《大数据经典论文解读》 课程介绍 >



讲述：王惠

时长 15:38 大小 14.33M

你好，我是黄金。今天这期复习课，咱们来回顾和总结思考下 Spanner 这篇论文的内容。

Spanner 介绍

Spanner，是第一个同时支持外部一致性分布式事务和全球部署的分布式数据库，它能够伸缩至百万台服务器、横跨数百个数据中心、存储万亿条记录。那么，在这么大规模的分布式系统中，**如何高效地支持外部一致性事务**，就是我们需要关注的重点。

课程内容回顾

我们先来回顾下徐老师所讲的课程内容。

第一讲谈了 Spanner 的架构与实现。在架构上，Spanner 由多个 Zone 构成，所有的 Zone 由一个 Universe 管理。Zone 负责读写数据，它的结构类似 Bigtable，由 Zonemaster、Spanserver 和 Location Proxy 构成；Universe 负责管理 Zone 的状态，在 Zone 与 Zone 之间调度数据。

在实现上，数据是通过 (key:string, timestamp:int64) -> value:string 这样的映射关系来表示的，TimeStamp，也就是版本是在整行数据上，而不是像 Bigtable 那样在列上。数据的组织顺序像 Megastore 的 EntityGroup，关联在一起的数据在 Spanner 中被称为目录。

多个目录构成一个 Tablet，Tablet 和它的所有副本构成了 Paxos Group。目录可以在 Tablet 之间调度，频繁共同访问的目录可以调度到在物理上相邻的位置。多个 Tablet 构成一个 Spanserver，Spanserver 上管理了事务。

第二讲谈了不可靠的时钟，主要由三个问题构成。

第一个问题，不同机器上的时间为什么会存在差异？这是因为机器的时间通过石英振荡器来计时，但是石英振荡器存在误差，一天的误差会在 1 秒上下。

第二个问题，这种差异对分布式事务会造成什么样的影响？不同服务器的时间是不同的，那么分布式事务提交时间应该用哪台服务器的时间，徐老师给我们分析了三种情况：

> 第一种情况，使用参与者的本地时间作为事务提交时间。这不符合事务的基本要求，因为属于同一事务的多处修改的时间戳，应该要一致。
>
> 第二种情况，使用协调者的时间作为事务提交时间。不同事务的协调者不同，不同协调者的本地时间不同，可能会出现后一个事务的查询时间早于前一个事务的提交时间，导致后一个事务无法读取前一个事务写入的数据。
>
> 第三种情况，使用协调者和所有参与者之中最大的时间，作为事务提交时间。这会遇到和情况二一样的问题。简单地讲，不管使用谁的时间作为事务提交时间，都存在问题。

第三个问题，Spanner 是如何解决时钟差异带来的问题的？解决这个问题的关键在于，写入数据时，提供全局单调递增的事务提交时间，读取数据时，获取比数据最近更新时间还要晚的时间。

Spanner 的解决方案核心有两点，一个是尽量缩短服务器之间的时钟差异，二个是等待差异时间完全过去再提交事务。缩短服务器之间的时钟差异，依靠原子钟和 GPS。等待差异时间完全过去再提交事务，依靠 TrueTime，这一点我会在后面详细解释。

第三讲解释了外部一致性的概念，介绍了 TrueTime API，以及 Spanner 的并发模型，包括读写事务、只读事务和快照读。所谓外部一致性，就是既要保证可串行化，又要保证可线性化。抛开这些概念，外部一致性也可以理解成既要保证事务看起来是顺序执行的，也要保证前一个事务写入的数据，后一个事务马上能读到。

读写事务、只读事务和快照读这三种操作，只有读写事务涉及写操作，也只有读写事务需要加锁。只读事务和快照读都是读操作，它们的区别在于只读事务的读时间戳由 Spanner 提供，快照读的读时间戳由应用程序提供。

关于读写事务，徐老师讲解了如何分配事务提交时间，如何保证分配的时间单调递增，以及读写事务的整个流程。关于只读事务，徐老师讲解了如何分配读时间戳 (s_read)，如何维护和推进副本的数据版本时间 (t_safe)。如果读时间戳 (s_read) 小于等于副本的数据版本时间 (t_safe)，副本可以直接返回结果，否则，副本需要等待 t_safe 推进，直到大于读时间戳 (s_read)，才能返回数据。

Spanner 论文的内容很多，我们主要会来复习三点内容。第一是在**架构与实现**上，关注 Spanner 和 Megastore 的异同；第二是在**写操作**上，关注如何分配全局单调递增的事务提交时间；第三是在**读操作**上，关注如何分配读时间戳 (t_read)，如何维护和推进副本的最新数据时间 (t_safe)。

架构与实现

粗略地来看，Spanner 的架构并没有什么特殊之处。整个数据库分成多个 Zone，每个 Zone 提供的功能很像 Bigtable，数据通过 Paxos 算法同步，数据模型和 Megastore 一样。这样的架构，性能能比 Megastore 强到哪里去？

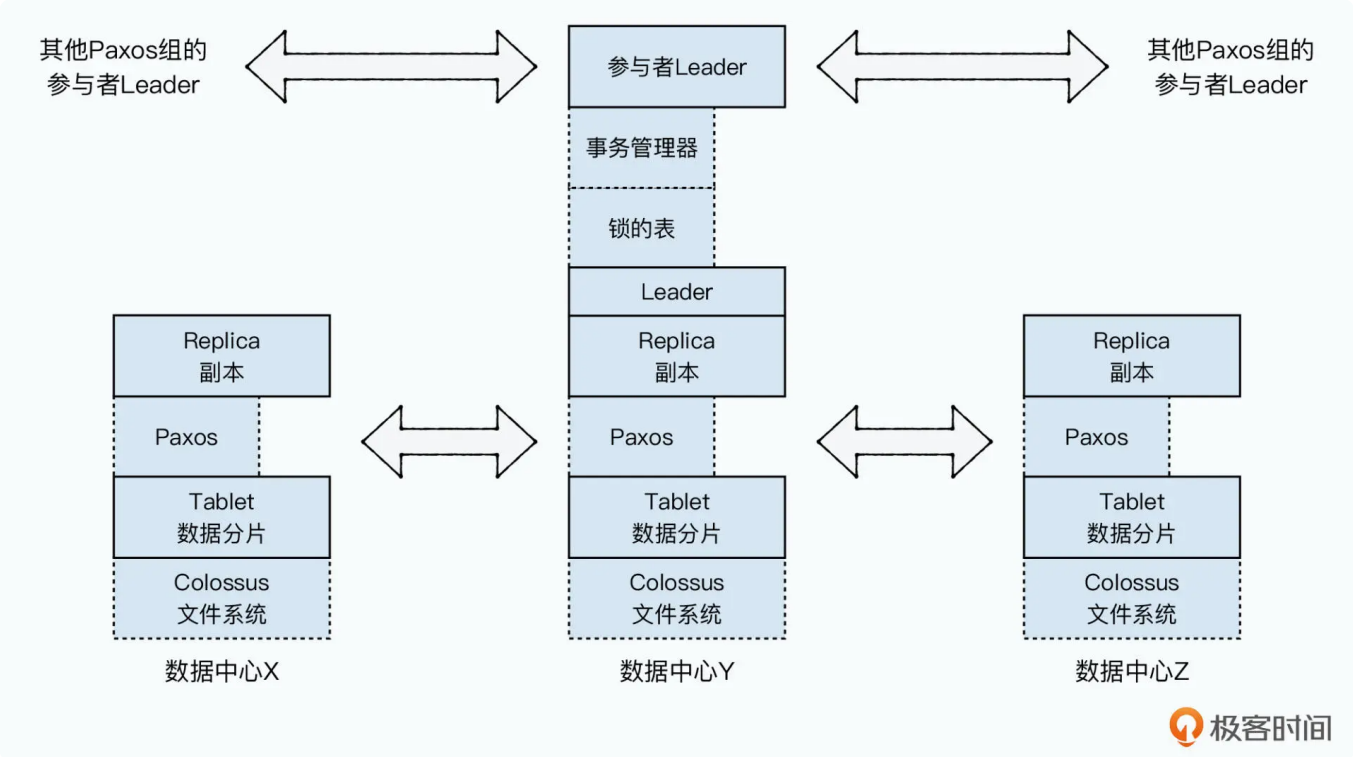
但凑近来看，Spanner 的实现又有很多不同，我把它总结成三点。

第一，智能调度数据，降低访问延迟。首先，用户可以控制数据分配策略。比如哪些数据中心包含哪些数据，数据距离用户有多远，数据副本之间的距离有多远，需要多少个数据副本等等。其次，不同目录会根据共同访问频率，自动调度到相同的 Tablet 上。

第二，放弃 Bigtable，重新实现 Spanserver。观察 Spanserver 软件栈这张图，可以发现 Spanserver 和 Bigtable 的两处区别。

首先，Tablet 的职责不同。在 Spanserver 中，Tablet 不仅负责存储数据，还担当了 Paxos Group 的节点。Tablet 之间同步数据，并不依赖额外分布式锁服务，而是自身实现了 Paxos 算法。我想这是因为 Tablet 的功能已经非常清晰，而且在 Spanner 的场景下，效率比简单要重要得多。

其次，Replica，对应着 Bigtable 的 TabletServer，它的职责也不同。它管理了锁表，承担了事务管理器的职责。如果使用 Bigtable，相当于在 Bigtable 内部扩展这些功能，而不是在 Bigtable 之上实现这些功能。



第三，Leader 有租期的 Pipelined Paxos 算法。我们知道，Megastore 的 Paxos Leader 没有租期，解决了 Leader 节点不可用，可以马上选举新 Leader 的问题。但是单个 EntityGroup 的频繁写请求，会导致 Paxos 提议退回两阶段请求，这在多数据中心的广域网中会造成很高的延迟。

在广域网中，更好的做法是像 Spanner 这样，当前面的提议没有完成时，后面的提议在 Leader 上排队，等前一个完成之后再开始，这样的提议只需要一阶段请求，这就是 Pipelined Paxos 算法。

全局单调递增的事务提交时间

Spanner 的一个关键特性是满足外部一致性的分布式事务。这个特性的关键在于哪怕多个事务发生在完全不同的服务器，也能够分配全局单调递增的事务提交时间戳。而获得全局单调递增的时间戳，关键又在于 TrueTime 的 API 和实现。

在 Spanner 论文的第二讲中，徐老师讲过，不管是选择协调者的本地时间，还是选择协调者和所有参与者之中最大的本地时间，作为分布式事务提交时间，都会遇到时间落后的参与者查不到最新数据的问题。解决的办法就是让协调者等待，等待所有参与者本地时间超过事务提交时间再提交事务，之后参与者再以本地时间查询，因为本地时间大于最近的事务提交时间，就能查到最新的数据。

那么，协调者怎么在不和参与者通信的情况下，知道参与者的本地时间超过了事务提交时间的呢？

答案就是 **TrueTime**。

TrueTime 定义了一个时间范围，表示当前所有服务器的时间范围。协调者只要等待事务提交时间小于 TrueTime 定义的时间范围，就可以认为所有参与者的本地时间超过了事务提交时间。

那么有没有可能出现先提交事务 T1，后提交事务 T2，但是 T2 的时间戳小于 T1 的时间戳的情况？

我们先看看**事务发生在单个 Paxos Group** 的情况。事务的提交时间由 Paxos Leader 分配，Leader 会保证后一个事务的提交时间大于前一个事务的提交时间。当 Leader 想分配一个比自己租期结束时间更大的事务提交时间，它需要先延长租期，延长至租期结束时间大于事务提交时间，才能分配出这个事务提交时间。

Leader 变更时，也通过 TrueTime API，保证了后一个 Leader 的租期开始时间大于前一个 Leader 的租期结束时间。所以，即使 Leader 变更，后面分配的事务提交时间也会大于之前的事务提交时间。

接着我们再看看**事务发生在多个 Paxos Group** 的情况。所有参与者的 Paxos Leader 分别选出各自的事务提交时间，这些 Paxos Leader 有一个会作为协调者，协调者会选出最

终的事务提交时间。

每个参与者保证选出的事务提交时间，比参与者之前应用的事务的提交时间都要大，协调者保证最终选出的事务提交时间，比所有参与者选出的事务提交时间都要大。所以事务发生在多个 Paxos Group，也可以保证事务提交时间戳单调递增。

读操作

你应该还记得 Megastore 实现快速读的方法，是通过协调者服务判断本地数据中心是否包含所读数据的最新版本，如果包含，则直接从本地数据中心返回结果，整个过程不需要任何跨数据中心的网络请求。

Spanner 为了快速响应读操作，也会尝试直接从本地副本返回数据，**那 Spanner 又是如何保证本地副本返回数据的安全性的呢？**

Spanner 的每个副本都会维护一个叫做 t_safe 的时间戳，只要读时间戳 s_read 小于等于 t_safe，副本就可以安全地返回数据。t_safe 和 s_read 相当于数据版本，s_read 小于等于 t_safe，意味着副本本地包含要读取的数据版本。

副本维护的 t_safe 是这样取值的，t_safe=min(t_paxos_safe, t_TM_safe)。其中 t_paxos_safe 是最近应用的 Paxos 写操作的时间戳，t_TM_safe 的值和事务管理器中是否存在正在进行中的事务有关，没有进行中的事务，t_TM_safe 等于无穷大，存在进行中的事务，t_TM_safe 要小于所有事务的 prepare 时间。

我不知道你有没有想过，**为什么 t_safe 会和 t_TM_safe 扯上关系？**

最初我觉得，让 t_safe 等于 t_paxos_safe 就够了，Paxos 写入的数据就是业务数据，所以 Paxos 写入数据的最新时间就是安全的读取时间。但还需要考虑 t_TM_safe，就说明 t_TM_safe 可能小于 t_paxos_safe，我猜想可能的原因是事务执行的过程信息也是通过 Paxos 写入，所以 t_paxos_safe 可能代表写入事务信息的时间戳，而不是业务数据的最后修改时间。

读操作都需要带上读时间戳 s_read，客户端可以指定 s_read，也可以让 Spanner 分配 s_read。对于只涉及单个 Paxos Group 的读请求，由 Paxos Leader 分配 s_read，Leader 会选择最后提交的写操作的时间戳。

对于涉及多个 Paxos Group 的读请求，一种方法是通过所有参与者的 Paxos Leader 选出最小的 s_read，不过论文中说，当年采用了一种更简单的方法，直接使用当前时间 TT.now().latest。

针对总是存在读操作的读时间戳 s_read 大于副本的安全时间 t_safe 的情况，这个时候读操作会被阻塞，直到 t_safe 推进到大于等于 s_read 的时间。继续在副本所属的 Paxos Group 上执行写操作，会推进 t_safe。没有写操作执行的时候，Paxos Leader 也会定时推进 t_safe。

小结

好了，Spanner 这篇论文我们就复习到这里。回到我们最初关注的重点，Spanner 作为一个超大规模的分布式系统，如何高效地支持外部一致性事务。支持外部一致性事务，需要全局单调递增的事务提交时间。生成和维持全局有序的时间，不是靠服务之间频繁的通信，而是靠把所有服务器的时钟误差维持在同一个范围内，这就是 TrueTime 的作用。

Spanner 论文是我和你一起复习的最后一篇论文，和你一起复习的日子是我在大数据领域成长最快的日子。三个多月的时间，我从一个门外汉，变成了对大数据发展历史、关键技术原理有深刻认识的内行，我相信你也一定发生了很大的变化。在剩下的时间里，我会继续和你一起学习徐老师的课程，研读老师推荐的每一篇论文，以及每一份资料。

上一篇 复习课（九）| Megastore

下一篇 32 | Raft（一）：不会背叛的信使

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。