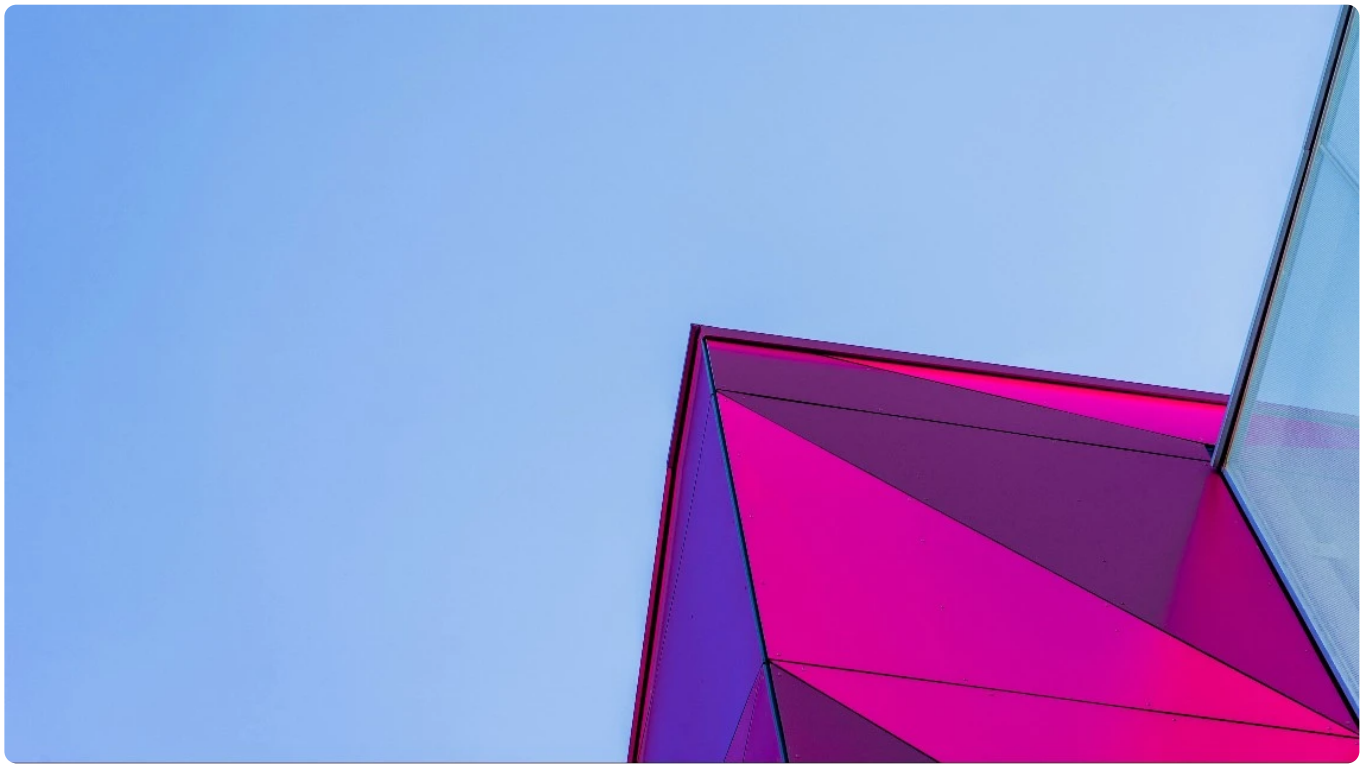
复习课（九）| Megastore

2021-12-20 黄金

《大数据经典论文解读》 课程介绍 >



讲述：王惠
时长 12:13 大小 11.19M

你好，我是黄金。今天我们来接着来复习下 Megastore 的论文。

Megastore 介绍

Megastore 是为了满足现代交互式在线服务的需求，而开发的存储系统。当今的交互式在线服务需要五种能力：

第一是**高度可伸缩**。随着在线服务的流行，用户的规模可能会增长到数百万，而用户规模的变化，就对服务的可伸缩性提出了要求。

第二是**可快速开发**。在线服务为了吸引用户，需要不断推出新功能，而新功能对数据存储模式的灵活性也提出了要求。

第三是**低延迟**。在线服务需要快速响应用户的请求。

第四是**数据的一致性视图**。数据修改之后，需要立即被用户看到，并且持久化存储起来。

第五是**高可用**。用户希望随时随地地访问在线服务，在线服务需要有足够的容错能力，在硬件故障、网络故障，甚至整个数据中心故障面前，依然能够提供服务。

而 Megastore，就正好混合了 NoSQL 数据库的灵活性和传统 RDBMS 的便利性。它既拥有 NoSQL 数据库那样的伸缩性、可用性和灵活的数据模式，又拥有 RDBMS 那样的 SQL API 和事务的特性。

徐老师用三节课，给我们讲解了 Megastore，分别对应了论文的第二部分、第三部分和第四部分。第一讲的主题是**可用性和伸缩性**，包括数据如何复制，如何分区，以及在物理上如何存储。第二讲的主题是 **Megastore 的主要特征**，包括数据模型、事务和并发控制。第三讲的主题是**跨数据中心同步复制的实现**，包括如何实现快速读和快速写、Megastore 整体架构、读写操作流程、协同服务器的作用和故障恢复。

可用性和伸缩性

我们先来复习下 Megastore 的可用性和伸缩性。Megastore 提供了跨数据中心的可用性，操作被封装成预写日志，通过 Paxos 算法同步写入多个数据中心。Paxos 算法既保证了操作被写入多数节点，又保证了多个操作被顺序写入。如果整个 Megastore 只有一个 Paxos 组，就意味着所有操作被顺序写入，系统的并发程度会很低。

于是，Megastore 把存储数据分成多个**分区**，每个分区的操作由一个 **Paxos 组**协调，以此来保证分区内的操作顺序执行、分区间的操作并发执行。

在 Megastore 中，数据分区被称为**实体组**（Entity Group），一个实体组包含多个类型不同但彼此关联的实体。比如在一个博客系统中，一个用户可以看成一个实体组，这个实体组由用户信息，以及用户发表的博客等内容构成，用户信息和博客就是实体组的不同实体类型。在这样一个系统中，同一用户发表不同博客是顺序执行的，不同用户发表博客是并发执行的。

Megastore 在分区内，也就是一个实体组里，支持一阶段数据库事务。但如果一个操作涉及多个实体组，那就得使用昂贵的两阶段事务，或者放弃跨实体组的事务，使用异步消息机制。

另外，在一个数据中心里，Megastore 是使用 Bigtable 来存储操作日志和实际数据。而为了降低延迟，提高吞吐量，Megastore 允许程序控制数据所在的位置，也就是存储在哪些数据中心。这些数据中心距离用户位置越近，彼此距离越近，操作延迟就越低。

Megastore 的主要特征

Megastore 有两个主要特征，一个是**数据模型**，一个是**事务保障**。

我们先来看看数据模型。Megastore 的数据模型非常成功，在 Google 广受欢迎，作为 Megastore 的继任者 Spanner，它后来也采用了这种数据模型。在第二讲中，徐老师详细讲解了 Megastore 的数据模型，以照片分享服务为例，描述了用户和照片两个实体，是如何构成一个实体组，实体组的数据在 Bigtable 中又是如何布局的。

本质上，这种数据模型是通过**实体的 Key**，把不同类型的实体数据预 Join 在一起，在物理上构成连续分布。一个实体组的数据存储在一块连续的区域，就能利用**数据的局部性**提高缓存效率，降低延迟。而这种存储方式，也非常适合用 Bigtable 来实现。

Megastore 的每一个实体组，相当于一个迷你数据库。在这个迷你数据库上，Megastore 支持可串行化的 ACID 语义。我们之前学习过，可串行化指的是从外部应用看，事务是一个一个顺序执行的。现代的关系型数据库，都是采用一种叫做 **MVCC**（Multiversion Concurrency Control），也就是多版本并发控制的机制，来实现可串行化。

这里的多版本，是指数据存在多个历史版本，包括已经提交的事务产生的版本和正在进行的事务产生的版本。而读操作，是指读取的版本是已提交的最新版本。

Bigtable 的数据采用的是**多版本存储**，更新操作不是修改原来的数据，而是增加一个新版本，每个版本对应了一个时间戳。那么利用 Bigtable 的这个特性，我们可以在 Megastore 中实现 MVCC 机制。把提交事务的时间戳，作为数据变更的时间戳，以保证提交事务的时间戳单调递增。这样的话，给定任意一个时间戳，只能读取到事务提交后的数据状态，而不会看到部分更新的中间状态。

Megastore 的事务保障是这样实现的。Megastore 的读写操作不会相互阻塞，它是采用乐观并发机制，产生冲突之后重试操作。

假设有两个事务，事务 A 对一条数据执行读操作，事务 B 对同一条数据执行写操作。两个事务并发执行，事务 A 读取数据时，读取的版本是已提交的最新版本。等到事务 A 提交时，如果发现之前读取的版本不再是最新版本，就说明期间有其他事务提交，修改了数据，在这里就是事务 B，那么事务 A 会退回去重新执行，直到读取的版本在事务提交时依然是最新版本，才能成功提交。

跨数据中心同步复制

最后我们来谈谈跨数据中心同步复制机制。这中间的挑战主要是在于**跨数据中心带来的高延迟**，而优化思路，是在于**尽可能减少跨数据中心的网络交互次数**。

在这期复习课中，我不会带着你去复习 Megastore 的整体架构和读写流程，而是想从几个问题出发，来一起理解下 Megastore**快速读和快速写**的本质。

如何实现快速读？

要实现快速读，最好是从本地数据中心返回结果。但是本地数据中心怎么知道，自己的数据是不是最新的呢？

对于分布式系统而言，真相由**多数节点**决定。本地数据中心的数据是不是最新，一般需要询问存储相同数据的其他数据中心，得到多数数据中心的认可，才能确认是最新的。不过，Megastore 是通过协调者服务，实现了在本地数据中心判断数据是否最新。

协调者服务怎么知道数据是否最新？

通过 Paxos 算法写入数据时，只要写入多数节点，数据就被视为写入成功，而任一节点并不知道自己是不是属于这多数节点之一。Megastore 也是通过 Paxos 算法写入数据，不过在收到多数数据中心的写入确认后，Megastore 并没有结束写入操作，而是**同步通知剩余的数据中心，它们的数据不是最新版本**。

通知的服务就是协调者服务，所以协调者服务知道，本地数据中心中的哪些数据是最新的、哪些数据是过期的。不过代价你也看到了，写请求需要同步发送给存储这份数据的所有数据中心，而不是多数数据中心。虽然协调者服务不需要写入数据，但是把请求发送给协调者本身，可能已经带来了很高的延迟。

命中快速读的概率大吗？

读取数据时，发现本地数据中心的数据正好是最新版本，就能通过快速读的流程返回数据，否则需要通过大量的跨数据中心的网络请求，来追赶版本。

本地数据中心是不是最新，首先取决于**协调者是否可用**，如果协调者不可用，本地数据中心会认为数据不是最新的。其次取决于**实体组的设计和业务特点**，如果写入数据的数据中心，同时也是查询对应数据的数据中心，那么命中快速读的概率就大。如果实体组会频繁地在不同数据中心写入，或者频繁地在不同数据中心读取，那么命中快速读的概率就小。

如何实现快速写？

跨数据中心的快速写入在于高效的 **Paxos 算法**。Megastore 并没有采用基于 Master 的 Paxos 算法，而是采用了**基于 Leader** 的 Paxos 算法。

这两者的共同点在于，都需要选取一个主节点来解决并发提议带来的冲突问题，把提议从两阶段请求变成一阶段请求，把两次跨数据中心的网络交互，变成一次跨数据中心的网络交互。而两者的区别是在于，基于 Master 的 Paxos 算法中的主节点，也就是 **Master，是有租期的**，基于 Leader 的 Paxos 算法中的主节点，也就是 **Leader，是没有租期的**。

在基于 Leader 的 Paxos 算法中，一个提议不仅包含了操作或数据，还包含了下一个 Leader 的提名。没有租期，就意味着当 Leader 出现故障时，不用等待租期失效，才开始选举新的 Leader，而是可以立即退回到两阶段请求，马上选举新的 Leader。这样选出的 Leader，通常所在的数据中心，就是离用户最近的数据中心。

快速写到底快不快？

论文第 4.10 小节 Production Metrics，提到在 Google 内部，已经部署了一百多个使用 Megastore 的生产应用，其中写操作的平均延迟在 100~400ms。由此可见，**Megastore 的写操作性能很低。**

我认为主要是有三个原因，造成了这种结果：第一，Megastore 是使用 Bigtable 作为底层存储的，它会存在大量多余的操作；第二，Megastore 这种基于 Leader 的 Paxos 算

法，一旦冲突就退回到两阶段请求，增加了网络延迟；第三，它的写操作，需要同步通知到管理相应数据的所有数据中心，而不是多数数据中心。

小结

Megastore 并不是一个高性能的产品，但是它做出了混合 NoSQL 数据库和传统 RDBMS 的示范，让人们感受到了分布式事务数据库，可以提供多么强大的特性。至于它的性能问题，我们在下一篇论文 Spanner 中，会看到是如何被一一解决的。

上一篇 复习课（八）| Resilient Distributed Datasets

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。