02｜强一致性：那么多数据一致性模型，究竟有啥不一样？

2020-08-12 王磊

分布式数据库30讲 进入课程 >

讲述：王磊
时长 21:52　大小 20.04M

你好，我是王磊，你也可以叫我 Ivan。

我们经常会听到说，分布式数据库的一个优势在于，它能够支持 NoSQL 做不到的强一致性。你怎么看待这件事儿呢？

显然，要来分析这个问题，我们首先得明白“强一致性”意味着什么。

我也问过很多身边的朋友，他们的答案都不太一样。有人说，只要使用了 Paxos 或者 Raft 算法，就可以实现强一致性；也有人说，根据 CAP 原理只能三选二，分区容忍性和高可用性又是必不可少的，所以分布式数据库是做不到强一致性的。可是，这些观点或多或少都是有问题的。

那么，今天我们就来讲讲什么是“强一致性”。

一直以来，在“分布式系统”和“数据库”这两个学科中，一致性（Consistency）都是重要概念，但它表达的内容却并不相同。

对于分布式系统而言，一致性是在探讨当系统内的一份逻辑数据存在多个物理的数据副本时，对其执行读写操作会产生什么样的结果，这也符合 CAP 理论对一致性的表述。

而在数据库领域，“一致性”与事务密切相关，又进一步细化到 ACID 四个方面。其中，I 所代表的隔离性（Isolation），是“一致性”的核心内容，研究的就是如何协调事务之间的冲突。

因此，当我们谈论分布式数据库的一致性时，实质上是在谈论**数据一致性**和**事务一致性**两个方面。这一点，从 Google Spanner 对其外部一致性（External Consistency）的论述中也可以得到佐证。

数据一致性

今天，我会先介绍数据一致性，下一讲中，我再为你讲解事务一致性以及它们之间的关系。

包括分布式数据库在内的分布式存储系统，为了避免设备与网络的不可靠带来的影响，通常会存储多个数据副本。逻辑上的一份数据同时存储在多个物理副本上，自然带来了数据一致性问题。

讨论数据一致性还有一个前提，就是同时存在读操作和写操作，否则也是没有意义的。把两个因素加在一起，就是多副本数据上的一组读写策略，被称为“一致性模型”（Consistency Model）。一致性模型数量很多，让人难以分辨。为了便于你理解，我先建立一个简单的分析框架。

这里，我要借用论文“The many faces of consistency”中的两个概念，状态一致性（State Consistency）和操作一致性（Operation Consistency）。不要慌，这不是新的一致性模型，它们只是观察数据一致性的两个视角。

状态一致性是指，数据所处的客观、实际状态所体现的一致性；

操作一致性是指，外部用户通过协议约定的操作，能够读取到的数据一致性。

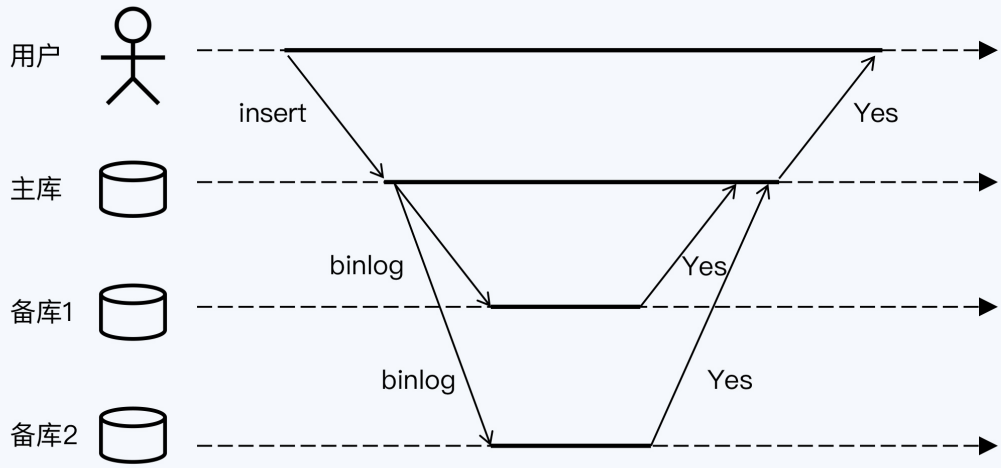
状态视角

从状态的视角来看，任何变更操作后，数据只有两种状态，所有副本一致或者不一致。在某些条件下，不一致的状态是暂时，还会转换到一致的状态，而那些永远不一致的情况几乎不会去讨论，所以习惯上大家会把不一致称为“弱一致”。相对的，一致就叫做“强一致”了。

下面，我以 MySQL 为例来说明状态视角的“强一致”。

强一致性：MySQL 全同步复制

现在有一个 MySQL 集群，由一主两备三个节点构成，那么在全同步复制（Fully Synchronous Replication）模式下，用户与 MySQL 交互的过程是这样的。



在该模式下，主库与备库同步 binlog 时，主库只有在收到两个备库的成功响应后，才能够向客户端反馈提交成功。

显然，用户获得响应时，主库和备库的数据副本已经达成一致，所以后续的读操作肯定是没有问题的，但这种模式的副作用非常大，体现在以下两点。

第一，**性能差**。主库必须等到两个备库均返回成功后，才能向用户反馈提交成功。图中由于网络阻塞，“备库 2”稍晚于“备库 1”返回响应，增加了数据库整体的延时。而下一

次，拖后腿的可能变成“备库 1”。总之，主库的响应时间取决于两个备库中延时最长的那个。

第二，**可用性问题**。我们在第 1 讲提到过可用性概念，任何设备都有可能出现故障，尤其是 x86 这样的通用商业设备，故障率会更高。但在全同步复制模式下，集群中的三个节点被串联起来，如果单机可用性是 95%，那么集群整体的可用性就是 85.7%（95%*95%*95%=85.7%），跟单机相比反而降低了。

集群规模越大，这些问题就越严重，所以全同步复制模式在生产系统中也很少使用。更进一步说，在工程实践中，实现状态视角的强一致性需要付出的代价太大，尤其是与可用性有无法回避的冲突，所以很多产品选择了状态视角的弱一致性。

弱一致性：NoSQL 最终一致性

NoSQL 产品是应用弱一致性的典型代表，但对弱一致性的接受仍然是有限度的，这就是 BASE 理论中的 E 所代表的最终一致性（Eventually Consistency），弱于最终一致性的产品就几乎没有了。

对于最终一致性，你可以这样理解：在主副本执行写操作并反馈成功时，不要求其他副本与主副本保持一致，但在经过一段时间后这些副本最终会追上主副本的进度，重新达到数据状态的一致。

你再仔细推敲一下，是不是觉得这个定义还有点含糊？“经过一段时间”到底是多久呢？几秒还是几分钟？如果是一个不确定的数值，怎么在工程中使用呢？

这就需要我们从操作视角来分析了。

操作视角

最终一致性，在语义上包含了很大的不确定性，所以很多时候并不是直接使用，而是加入一些限定条件，也就衍生出了若干种一致性模型。因为它们是在副本不一致的情况下，进行操作层面的封装来对外表现数据的状态，所以都可以纳入操作视角。

接下来，我会挑选 5 个常见的一致性模型逐一讲解。

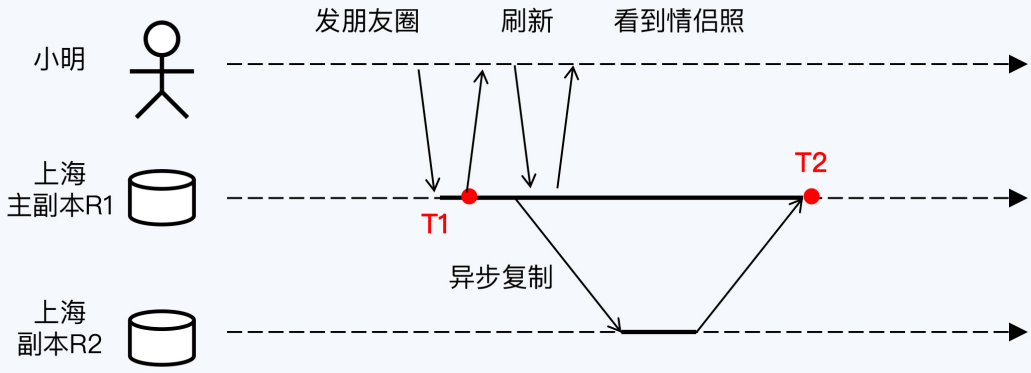
写后读一致性

首先来说**“写后读一致性”**（Read after Write Consistency），它也称为“读写一致性”，或“读自己写一致性”（Read My Writes Consistency）。你可能觉得最后一个名字听上去有些奇怪，但它却最准确地描述了这种一致性模型的使用效果。

我还是用一个例子来说明。

小明很喜欢在朋友圈分享自己的生活。这天是小明和女友小红的相识纪念日，小明特意在朋友圈分享了一张两人的情侣照。小明知道小红会很在意，特意又刷新了一下朋友圈，确认照片分享成功。

你是否意识到这个过程中系统已经实现了“写后读一致性”？我画了张流程图来表示这个过程。



小明发布照片的延时极短，用户体验很好。这是因为数据仅被保存在主副本 R1 上，就立即反馈保存成功。而其他副本在后台异步更新，由于网络的关系每个副本更新速度不同，在 T2 时刻上海的两个副本达成一致。从过程来看，这与前面所说的“最终一致性”完全相符。

要特别注意的是，小明有一个再次刷新朋友圈的动作，这时如果访问副本 R2，由于其尚未完成同步，情侣照将会消失，小明就会觉得自己的照片被弄丢了。此处，我们假定系统可以通过某种策略由写入节点的主副本 R1 负责后续的读取操作，这样就实现了写后读一致性，可以保证小明再次读取到照片。

自己写入成功的任何数据，下一刻一定能读取到，其内容保证与自己最后一次写入完全一致，这就是“读自己写一致性”名字的由来。当然，从旁观者角度看，可以称为“读你写

一致性”（Read Your Writes Consistency），有些论文确实采用了这个名称。

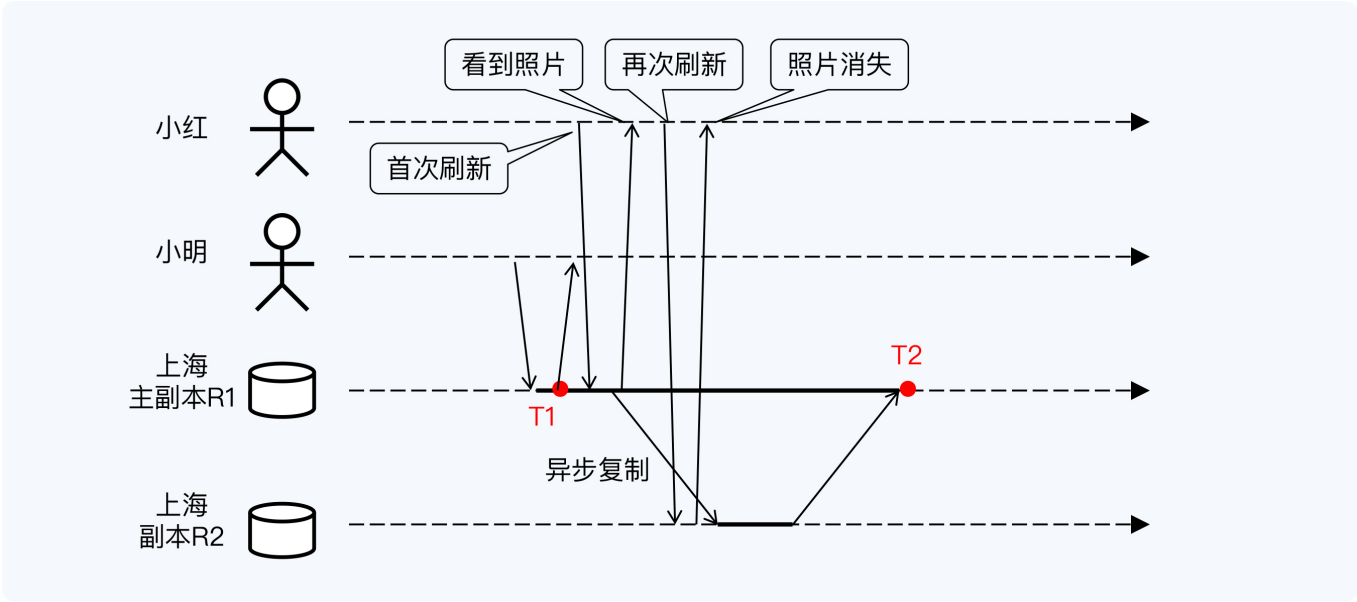
单调读一致性

但是，小明发完朋友圈之后，小红一定能看到照片吗？会不会发生异常呢？

这次确实出问题了。

此时，小红也在刷朋友圈，看到了小明刚刚分享的照片，非常开心。然后，小红收到一条信息，简单回复了一下，又回到朋友圈再次刷新，发现照片竟然不见了！小红很生气，打电话质问小明，为什么这么快就把照片删掉？小明听了一脸蒙，心想我没有删除呀。

你猜这中间发生了什么呢？我用另一张流程图来演示这种异常。



在小明发布照片后的瞬间，小红也刷新了朋友圈，此时读取到副本 R1，所以小红看到了照片；片刻之后，小红再次刷新，此时读取到的副本是 R2，于是照片消失了。小红以为小明删除了照片，但实际上这完全是程序错误造成的，数据向后回滚，出现了“时光倒流”。

想要排除这种异常，系统必须实现**单调读一致性**（Monotonic Read Consistency）。关于单调读一致性的定义，常见的解释是这样的：一个用户一旦读到某个值，不会读到比这个值更旧的值。

是不是感觉有点蒙？让我来解释一下。

假如，变量 X 被赋值三次，依次是 10、20、30；之后读取变量 X，如果第一次读到了 20，那下一次只有读到 20 或 30 才是合理的。因为在第一次读到 20 的一刻，意味着 10 已经是过期数据，没有意义了。

实现单调读一致性的方式，可以是将用户与副本建立固定的映射关系，比如使用哈希算法将用户 ID 映射到固定副本上，这样避免了在多个副本中切换，也就不会出现上面的异常了。

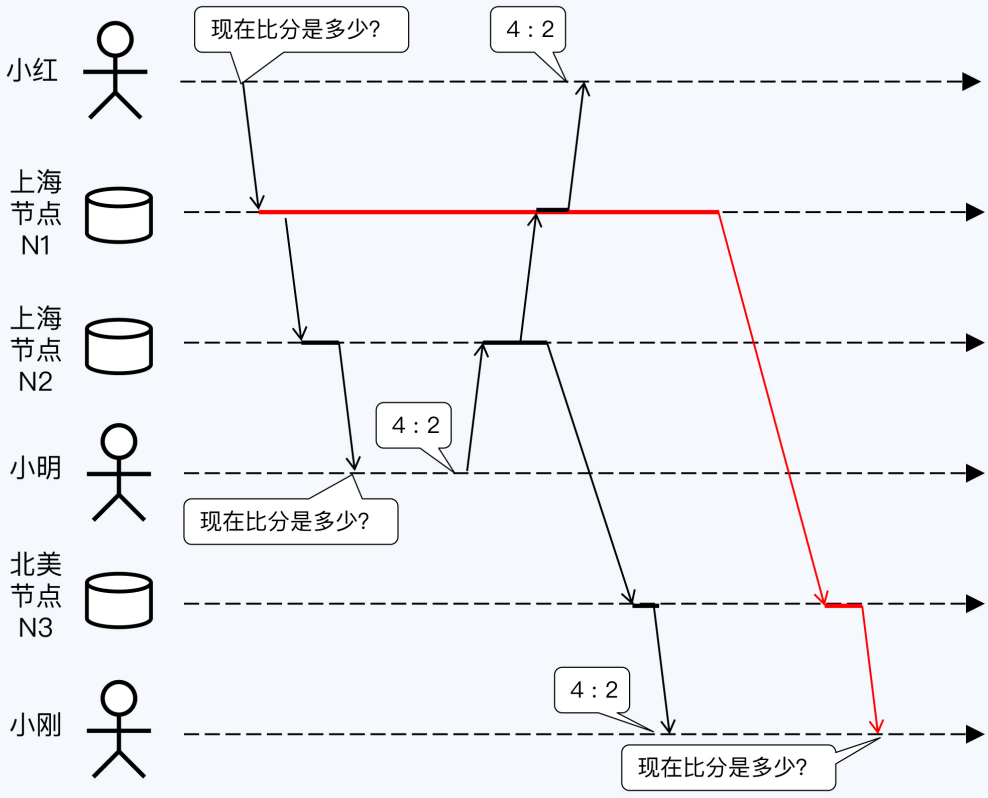
前缀一致性

但是，在一些更复杂的场景下还是会出现时间的扭曲。我再用一个例子来说明。

这天小明去看 CBA 总决赛，刚开球小明就拍了一张现场照片发到朋友圈，想要炫耀一下。小红也很喜欢篮球，但临时有事没有去现场，就在评论区问小明：“现在比分是多少？”小明回复：“4:2。”

小明的同学，远在加拿大的小刚，却看到了一个奇怪的现象，评论区先出现了小明的回复“4:2。”，而后才刷到小红的评论“现在比分是多少？”。难道小明能够预知未来吗？

这是什么原因呢？我们还是看图说话。



小明和小红的评论分别写入了节点 N1 和 N2，但是它们与 N3 同步数据时，由于网络传输的问题，N3 节点接收数据的顺序与数据写入的顺序并不一致，所以小刚是先看到答案后看到问题。

显然，问题与答案之间是有因果关系的，但这种关系在复制的过程中被忽略了，于是出现了异常。

保持这种因果关系的一致性，被称为**前缀读**或**前缀一致性**（Consistent Prefix）。要实现这种一致性，可以考虑在原有的评论数据上增加一种显式的因果关系，这样系统可以据此控制在其他进程的读取顺序。

线性一致性

在“前缀一致性”的案例中，问题与答案之间存在一种显式声明，但在现实中，多数场景的因果关系更加复杂，也不可能要求全部做显式声明。

比如对于分布式数据库来说，它无法要求应用系统在每次变更操作时附带声明一下，这次变更是因为读取了哪些数据而导致的。

那么，在显式声明无法奏效的情况下，如何寻找因果关系呢？

不知道你有没有听过这句话，“你所经历的一切，造就了现在的你。”是不是有一点哲学的味道？一切对原因的推测都是主观的，之前发生的一切都可能是原因。

所以，更可靠的方式是将自然语意的因果关系转变为事件发生的先后顺序。

线性一致性（Linearizability）就是建立在事件的先后顺序之上的。在线性一致性下，整个系统表现得好像只有一个副本，所有操作被记录在一条时间线上，并且被原子化，这样任意两个事件都可以比较先后顺序。

这些事件一起构成的集合，在数学上称为具有“全序关系”的集合，而“全序”也称为“线性序”。我想，线性一致性大概就是因此得名。

但是，集群中的各个节点不能做到真正的时钟同步，这样节点有各自的时间线。那么，如何将操作记录在一条时间线上呢？这就需要一个绝对时间，也就是**全局时钟**。

从产品层面看，主流分布式数据库大多以实现线性一致性为目标，在设计之初或演进过程中纷纷引入了全局时钟，比如 Spanner、TiDB、OceanBase、GoldenDB 和巨杉等等。

工程实现上，多数产品采用单点授时（TSO），也就是从一台时间服务器获取时间，同时配有高可靠设计； 而 Spanner 以全球化部署为目标，因为 TSO 有部署范围上的限制，所以 Spanner 的实现方式是通过 GPS 和原子钟实现的全局时钟，也就是 True Time，它可以保证在全球范围内任意节点能同时获得的一个绝对时间，误差在 7 毫秒以内。

但是，对于线性一致性，学术界其实是有争议的。反对者的论据来自爱因斯坦的相对论的一个重要结论，“时间是相对的”。没有绝对时间，也就不存在全序的事件顺序，不同的观察者可能对于哪个事件先发生是无法达成一致的。因此，线性一致性是有局限性的。

当然，从工程角度看，因为我们的应用场景都在经典物理学适用范围内，所以线性一致性也是适用的。

因果一致性

既然线性一致性不够完美，那么有没有不依赖绝对时间的方法呢？

当然是有的，这就是**因果一致性**（Causal Consistency）。

因果一致性的基础是**偏序关系**，也就是说，部分事件顺序是可以比较的。至少一个节点内部的事件是可以排序的，依靠节点的本地时钟就行了；节点间如果发生通讯，则参与通讯的两个事件也是可以排序的，接收方的事件一定晚于调用方的事件。

基于这种偏序关系，Leslie Lamport 在论文“Time, Clocks, and the Ordering of Events in a Distributed System”中提出了**逻辑时钟**的概念。

借助逻辑时钟仍然可以建立全序关系，当然这个全序关系是不够精确的。因为如果两个事件并不相关，那么逻辑时钟给出的大小关系是没有意义的。

多数观点认为，因果一致性弱于线性一致性，但在并发性能上具有优势，也足以处理多数的异常现象，所以因果一致性也在工业界得到了应用。

具体到分布式数据库领域，CockroachDB 和 YugabyteDB 都在设计中采用了**逻辑混合时钟**（Hybrid Logical Clocks），这个方案源自 Lamport 的逻辑时钟，也取得了不错的效果。因此，这两个产品都没有实现线性一致性，而是接近于因果一致性，其中 CockroachDB 将自己的一致性模型称为“No Stale Reads”。

时间对于任何一种分布式系统来说都是非常重要的，在分布式数据库中还会牵扯到数据一致性以外的很多话题，所以有关时间、全局时钟和逻辑时钟的内容，我还会在后续课程中提到并作详细讨论。

小结

好了，今天的内容就到这里。我们一起学习了数据一致性，希望你能够记住以下几点：

1. 一致性模型林林总总，数量繁多，但我们总可以从状态和操作这两个视角来观察，进而梳理出其读写操作的不同策略。
2. 从状态视角看，数据一致性只有两种状态，强一致或弱一致，而在实际系统中强一致是非常少见的，最终一致性是弱一致性的特殊形式；
3. 从操作视角看，最终一致性可以被封装成多种一致性模型，甚至是最强的线性一致性。

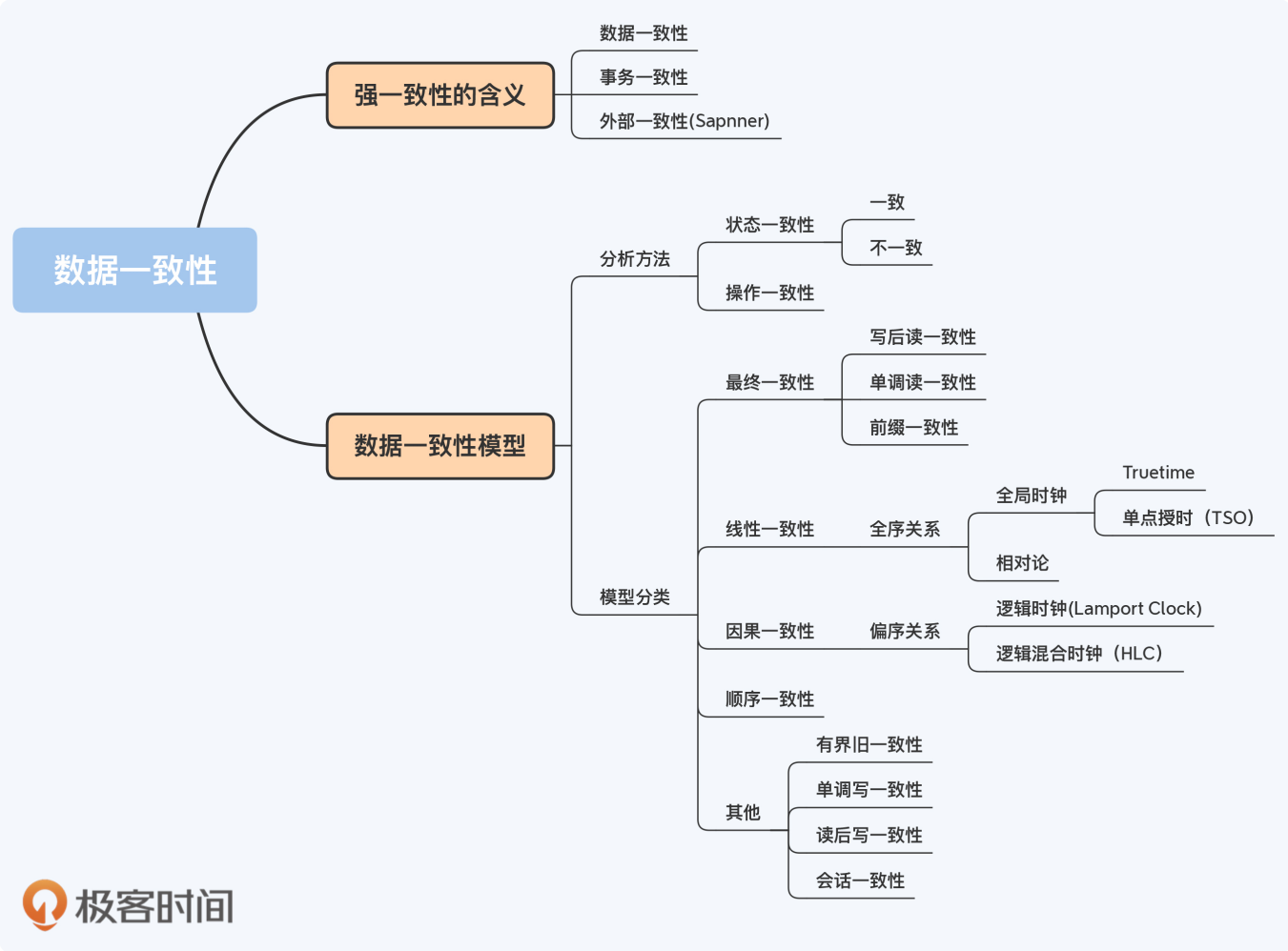
4. 分布式数据库主要应用了线性一致性或因果一致性。线性一致性必须要有全局时钟，全局时钟可能来自授时服务器或者特殊物理设备（如原子钟），全局时钟的实现方式会影响到集群的部署范围；因果一致性可以通过逻辑时钟实现，不依赖于硬件，不会限制集群的部署范围。

今天介绍的几种一致性模型，用一致性强度来衡量的话：线性一致性强于因果一致性；而写后读一致性、单调读一致性、前缀一致性弱于前两者，但这三者之间无法比较强弱。还有一种常被提及的顺序一致性（Sequentially Consistent），其强度介于线性一致性与因果一致性之间，由于较少在分布式数据库中使用，所以并没有介绍。

综上所述，我们提到的一致性模型强度排序如下：

线性一致性 > 顺序一致性 > 因果一致性 > { 写后读一致性，单调一致性，前缀一致性 }

此外，还有一些常见的弱一致性模型今天并没有提到，包括有限旧一致性（Bounded Staleness）、会话一致性（Session Consistency）、单调写一致性（Monotonic Write Consistency）和读后写一致性（Write Follows Read Consistency）等。如果你感兴趣，可以在 Azure Cosmos DB 的官方文档找到非常详细的说明。



思考题

课程的最后，我要给你留一道思考题。我们今天集中讨论了数据一致性，但是并没有特别强调 Paxos 的作用。这等于是说，Paxos 不是实现强一致性的必要条件。可是，有些时候大家又会将 Paxos 称为一致性协议。你觉得这个“一致性协议”和数据一致性又是什么关系呢？

欢迎你在评论区留言和我一起讨论，我会在答疑篇回复这个问题。最后，谢谢你的收听，如果你身边的朋友也对强一致性或者数据一致性这个话题感兴趣，欢迎你把今天这一讲分享给他，我们一起讨论。

提建议

上一篇 01 | 什么是分布式数据库？

下一篇 03 | 强一致性：别再用BASE做借口，来看看什么是真正的事务一致性

精选留言 (12)

写留言

峰
2020-08-12

感觉很长一段时间都被翻译给耽误了，ACID的C是一致性，强调的是数据的状态变迁的特性，CAP里的C共识，强调的是多副本条件下，多个节点怎么就数据的变动，达成共识，统一修改。
而paxos，raft是在牺牲一定A的条件下（多数节点存活才ok），实现C的一种多节点的通信协议，Paxos貌似不需要主节点这个角色去统一时序，Raft，zab需要主节点，它们都...
展开

作者回复: 你好，其实CAP的C也是Consistency，是多副本、单操作的数据一致性；而ACID里的C是指单副本、多操作的事务一致性。Paxos这类共识算法，可以看作是复制协议的一种，虽然有时也叫做一致性协议，但这个一致性是指Consensus。Consensus是实现数据一致性目标下的具体技术，但并不是唯一的选择。采用主从复制也可以达到同样效果，比如04讲会提到的PGXC风格的分布式数据库就是采用主从复制的方式。

3 7

tt
2020-08-13

我觉得数据一致性是从数据的用户视角出发对数据属性的描述，而paxos协议是达成共识的过程的一种实现方式，是从数据的生产者或者维护者角度出发的
展开

作者回复: 说的很好

2

扩散性百万咸面包
2020-08-12

Paxos本质上是共识算法，主要是用来维护数据库副本的一致性/权威性。而今天讲的一致

性是从用户角度来谈，而不局限于是数据副本。
同时，今天讲的一致性也需要共识算法Paxos，Raft来保证。比如选举，如何才能选出正确的Leader等等。

展开

作者回复: 高级别的一致性模型，可以基于Raft算法复制，但使用主从复制也是可以的😊

2

叫我皮卡丘
2020-08-12

我认为是数据的一致性依靠paxos,raft等一致性算法来保证

作者回复: 你好，即使是Raft协议，如果开放follower读，也会出现不一致的情况，所以读写策略还是很重要的。

2

南国
2020-08-13

先回答问题，“一致性协议”和数据一致性的关系是什么？很多留言的朋友都提到了一个重要的问题，即Paxos是一个共识算法，共识算法就是全局节点就某一事实达成一致，而数据一致中的数据我觉得可以理解为共识算法的日志，从此看来数据一致就是一致性协议的一个子集；而共识算法还包括很多其他部分，比如容错，日志压缩，集群变更等。
还有就是这些共识算法基本都遵从quorum的，所以都可以看成操作一致性，这也是用...

展开

作者回复: 你好，CAP中的C就是Consistency，是数据一致性，也是我们所说的操作视角的一致性，这里包含的多副本和读写策略两层含义。共识算法是复制协议层面的内容，并不一定对操作做严格定义。比如，就算我们使用Raft算法，但是如果开放了Follower读，也有可能达不到线性一致性或因果一致性的。事实上，CockroachDB的Follower读就是这样的。

1

chenchukun
2020-08-13

从状态和操作两个视角看待副本的一致性这点讲的很透彻，之前都没有考虑过这点。
从状态视角看，是不是只有全同步这种方式实现了强一致性，即使像paxos、raft这些实现了操作上线性一致性的算法，从状态视角看也不是强一致的。

然而全同步降低了系统的可用性，paxos、raft不保证所有节点状态的一致，而是通过额外的算法来保证操作视角的一致性，同时提高了系统的可用性。

展开

作者回复: 你好，你的理解非常准确，点赞

1

扩散性百万咸面包

2020-08-12

强一致性和弱一致性的定义感觉还是不够准确。
1. MySQL这个例子是全同步复制，实际上Raft也是强一致性算法，但它在应答客户端的请求成功后并不保证多副本之间暂时的数据一致性，有可能数据存在不同。只不过在收到读请求的时候会转发给Master，保证强一致性。
...

展开

作者回复: 你好，关于第一点，我再补充一下。
Raft是多数派协议，从写入成功那一刻的数据状态来说，肯定不是一致的。不过，通过操作方面的封装，约定由主副本对外提供服务，所以不会体现出副本间的差异。一致性模型，除了副本的状态，还要看读写操作。最终一致性的定义，其实只是描述了副本的状态而已。我认为，一致性模型，主要还是从读写操作的效果来分析，也数据副本的一致性有关但不是强依赖。比如，如果不使用Raft，用半同步，也可以做到线性一致性。
第二点，我没有完全理解，咱们可以继续探讨

1 1

孟磊

2020-08-17

和那些偏理论的课程不同，能感觉到作者对于分布式数据库的理解非常深刻，且结合了实际的金融业务，有点追剧的感觉了。 能不能拿出OceanBase goldendb这类领头羊产品给大家讲讲选型要注意的？

作者回复: 你好，孟磊，谢谢你的鼓励。我在构思这个专栏的时候，就订下一个目标，就是把学术的东西和目前工业界的实践联系起来，再落到具体的工作中，比如技术选型。所以，能得到你的肯定，我很高兴。当然，对产品的关注是必不可少的，从04开始的每一讲我都会对领头羊产品做局部设计上的拆解，并且比对不同方案的优劣，不过这个领头羊并不固定，因为我想向你介绍最有特点的设计。希望你能喜欢这种组织方式，后面的课程中，期待还能收到你的反馈，我们结合问题一起讨论。

南国
2020-08-15

老师，其实我还是没太懂前缀一致性和因果一致性的区别，前缀一致性是某些关系可比，并发的不可比，不也是一个偏序关系嘛？我还一直觉得这两个是一回事呢。

展开

作者回复: 你好，简单的说，因果一致性是靠逻辑时钟确定偏序关系，不需要应用介入；而前缀一致性靠事件之间显式声明的依赖关系，可以在应用层处理

南国
2020-08-14

老师，想问问会话一致性中会话怎么理解呢？

展开

作者回复: 你好，会话一种致性的会话就是通常所指的用户Session，它是多种一致性模型的组合，可以参考课程中Cosmos DB的官方文档学习。

南国
2020-08-14

前缀一致性和因果一致性有什么区别呢？看起来都是在描述一个因果的关系啊

作者回复: 你好，要实现前缀一致性，只要显示声明依赖关系就可以，这个有很多灵活的做法。而因果一致性，建立在偏序关系的基础上，很难在应用层面实现，要有底层的支持。

hql
2020-08-12

paxos协议定义的是一种决策过程。课程里的一致性是客观定义。

作者回复: 是的，一致性模型里有两个要点，读写策略和多副本状态。