

26 | 一致性与共识（一）：数据一致性都有哪些级别？

2022-04-01 陈现麟

《深入浅出分布式技术原理》

课程介绍 >



讲述：张浩

时长 11:53 大小 10.90M



你好，我是陈现麟。

通过学习“事务”序列的内容，我们从事务的四个特性 **ACID** 的角度讨论了相关的知识与技术原理，这样在以后的工作中，事务对我们来说就不再是一个陌生和难懂的概念，而是越发清晰了。我们能清楚地知道事务能提供哪些保障，我们的代码逻辑可能会出现什么样的异常情况，以及怎么避免这些异常情况的出现。恭喜你在学习分布式的道路上又前进了一大步！

不过，在前面课程的学习中，我们经常会碰到两个概念：多副本数据的一致性和多节点的共识，比如在分布式锁、事务的原子性等场景中。其实在分布式系统中，一致性和共识是两个绕不过的话题，现在各种各样的分布式系统都是建立在一致性和共识之上的，可以说**没有一致性和共识，就没有可用的分布式系统**。

既然一致性和共识对于分布式系统来说这么关键，那么我们一定要好好掌握。可是，通过前面课程中对一致性和共识场景的讨论，你现在虽然对二者有了很多的感性认识，知道在什么场景

下会遇到一致性和共识方面的问题，也知道一些具体的解决方案，但是如果要你具体介绍一致性和共识的话，心里是不是不太有底呢？

所以，从这节课开始，我们将一起花三节课的时间来解决这个问题。这一节课，我们先介绍一致性问题的来源，然后我们从一致性模型从强到弱的角度，来介绍几种经典的一致性的模型，并且一起讨论和对比各个一致性模型之间的差异。

一致性问题的来源

虽然数据一致性是分布式系统的基石，但是其实最早研究一致性的场景并不是分布式系统，而是多路处理器。不过我们可以将多路处理器理解为单机计算机系统内部的分布式场景，它有多个执行单元，每一个执行单元都有自己的存储（缓存），一个执行单元修改了自己存储中的一个数据后，这个数据在其他执行单元里面的副本就面临数据一致的问题。

当时间走到 1990 年代时，由于互联网公司的快速发展，单机系统在计算和存储方面都面临瓶颈，分布式是一个必然的选择，但是这也进一步放大了数据一致性面临的问题。对于数据的一致性，最理想的模型当然是表现得和一份数据完全一样，修改没有延迟，即所有的数据修改后立即被同步，但是这在现实世界中，数据的传播是需要时间的，所以**理想的一致性模型是不存在的**。

不过从应用层的角度来看，我们并不需要理想的一致性模型，只需要一致性模型能满足业务场景的需求就足够了，比如在一些统计点赞数的场景中，是能容忍一定的误差的，而评论之类的场景中，可能只要有因果关系的操作顺序一致就可以了。

同时由于一致性要求越高，实现的难度和性能消耗就越大，所以我们**可以通过评估业务场景来降低数据一致性的要求**，这样人们就定义了不同的一致性模型来满足不同的需求。是不是发现了这里的思考逻辑和事务的隔离级别一样了？都是正确性和性能之前的衡权。

讨论完了一致性问题的来源后，接下来我们从客户端读写操作的维度来讨论一致性模型。由于一致性模型的定义大多是基于数学语言来定义的，理解起来有一定的难度，所以在课程中，我们尽量用简单的语言来讨论。

接下来，我们将讨论四个经典且常见的一致性模型：线性一致性、顺序一致性、因果一致性和最终一致性。

线性一致性

线性一致性模型（Linearizability）是 Herlihy 和 Wing 等于 1987 年在论文 “Axioms for Concurrent Objects” 中提出的，线性一致性也被称为原子一致性（Atomic Consistency）、强一致性（Strong Consistency）、立即一致性（Immediate Consistency）和外部一致性（External Consistency）。

线性一致性是非常重要的一个一致性模型，在分布性锁、Leader 选举、唯一性约束等很多场景都可以看到它的身影。对于线性一致性的描述，我们可以从读写操作的维度来描述。

对于写操作来说，任意两个写操作 x_1 和 x_2 ：

- 如果写 x_1 操作和写 x_2 操作有重叠，那么可能 x_1 覆盖 x_2 ，也可能 x_2 覆盖 x_1 ；
- 如果写 x_1 操作在写 x_2 开始前完成，那么 x_2 一定覆盖 x_1 。

对于读操作来说：

- 写操作完成后，所有的客户端都能立即观察到；
- 对于多个客户端来说，必须读取到一样的顺序。

我们可以看到，线性一致性保证了所有的读取都可以读到最新写入的值，即一旦新的值被写入或读取，所有后续的读都会看到写入的值，直到它被再次覆盖。**在线性一致性模型中不论是数据的覆盖顺序还是读取顺序，都是按时间线从旧值向新值移动，而不会出现旧值反转的情况。**

顺序一致性

顺序一致性模型（Sequential Consistency）是 Leslie Lamport 在 1979 年发表的论文 “How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Program” 中提出的，在论文中具体的定义如下：

A multiprocessor is said to be sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

如果任何执行的结果与所有处理器的操作都以某种顺序执行的结果相同，并且每个单独的处理器的操作按照其程序顺序出现在该序列中，则称多处理器是顺序一致的。

对于顺序一致性，论文中的定义虽然严谨，但是理解起来也是有难度的，它需要掌握一些前置的定义，比如“**program order**”。不过在这里，我们依然可以用简单的语言来描述。

对于写操作来说，任意两个写操作 **x1** 和 **x2**：

- 如果写 **x1** 操作和写 **x2** 操作有重叠，那么可能 **x1** 覆盖 **x2**，也可能 **x2** 覆盖 **x1**；
- 当写 **x1** 操作在写 **x2** 开始前完成，如果两个写操作没有因果关系，当写 **x1** 操作在写 **x2** 开始前完成，那么有可能 **x1** 覆盖 **x2**，也有可能 **x2** 覆盖 **x1**；如果两个写操作有因果关系，即同一台机器节点先写 **x1**，或者先看到 **x1** 然后再写 **x2**，则所有节点必须用 **x2** 覆盖 **x1**。

对于读操作来说：

- 如果写操作 **x2** 覆盖 **x1** 完成，那么如果一个客户端到 **x2** 后，它就无法读取到 **x1** 了，但是这个时候，其他的客户端还可以观察到 **x1**；
- 对于多个客户端来说，必须观察到一样的顺序。

相对于线性一致性来说，**顺序一致性在一致性方面有两点放松**：

- 对于写操作，对没有因果关系的非并发写入操作，不要求严格按时间排序；
- 对于读操作，只要求所有的客户端观察到的顺序一致性，不要求写入后，所有的客户端都必须读取新值。

因果一致性

因果一致性模型（Causal Consistency）是 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, Phillip W. Hutto 在 1991 年发表的论文 “Causal memory: definitions, implementation, and programming” 中提出的一种一致性强度低于顺序一致性的模型。在这里，我们依然从读写操作的维度来进行描述。

对于写操作来说，任意两个写操作 **x1** 和 **x2**：

- 如果两个写操作没有因果关系，那么写 **x1** 操作在写 **x2** 开始前完成，有的节点是 **x1** 覆盖 **x2**，有的节点则 **x2** 可能覆盖 **x1**；
- 如果两个写操作有因果关系，即同一台机器节点先写 **x1**，或者先看到 **x1** 然后再写 **x2**，则所有节点必须用 **x2** 覆盖 **x1**。

对于读操作来说：

- 如果写操作 **x2** 覆盖 **x1** 完成，那么如果一个客户端到 **x2** 后，它就无法读取到 **x1** 了，但是这个时候，其他的客户端还可以观察到 **x1**。

相对于顺序一致性来说，**因果一致性在一致性方面有两点放松：**

- 对于写操作，对没有因果关系的非并发写入操作，不仅不要求按时间排序，还不再要求节点之间的写入顺序一致了；
- 对于读操作，由于对非并发写入顺序不再要求一致性，所以自然也无法要求多个客户端必须观察到一样的顺序。

最终一致性

最终一致性模型（Eventual Consistency）是 Amazon 的 CTO Werner Vogels 在 2009 年发表的一篇论文“Eventual Consistency”里提出的，它是 **Amazon 基于 Dynamo 等系统的实战经验所总结的一种很务实的实现**，它不同于前面几种由大学计算机科学的教授提出的一致性模型，所以也没有非常学院派清晰的定义，但是我们依然可以从读写操作的维度来描述它。

对于同一台机器的两个写操作 **x1** 和 **x2** 来说：

- 如果写 **x1** 操作在写 **x2** 开始前完成，那么所有节点在最终某时间点后，都会用 **x2** 覆盖 **x1**。

对于读操作来说：

- 在数据达到最终一致性的过程中，客户端的多次观察可以看到的结果是 **x1** 和 **x2** 中的任意值；
- 在数据达到最终一致性的过程后，所有客户端都将只能观察到 **x2**。

我们可以看出来，“最终”是一个模糊的、不确定的概念，它是没有明确上限的，Vogels 提出这个不一致的时间窗口可能是由通信延迟、负载和复制次数造成的，但是最终所有进程的观点都一致，这个不一致的时间窗口可能是几秒也可能是几天。

所以，**最终一致性是一个一致性非常低的模型**，但是它能非常高性能地实现，在一些业务量非常大，但是对一致性要求不高的场景，是非常推荐使用的。

总结

到这里，我们已经讨论完了几种最经典也最常见的一致性模型，现在我们来对这节课的内容做一个总结。

首先，我们讨论了一致性问题最早出现在多路处理器的场景，现在在分布式系统中广泛出现。同时，我们还得出了一个结论：对一致性模型进行分级是正确性和性能之间的一个权衡。

接着，我们从一致性模型强弱的维度，讨论了四个经典一致性模型的定义与差异，这里我们再从其他的维度描述一下，让你对一致性模型有一个更立体的理解。

第一，现在可以实现的一致性级别最强的是线性一致性，它是指所有进程看到的事件历史一致有序，并符合时间先后顺序, 单个进程遵守 **program order**，并且有 **total order**。

第二，是顺序一致性，它是指所有进程看到的事件历史一致有序，但不需要符合时间先后顺序, 单个进程遵守 **program order**，也有 **total order**。

第三，是因果一致性，它是指所有进程看到的因果事件历史一致有序，单个进程遵守 **program order**，不对没有因果关系的并发排序。

第四，是最终一致性，它是指所有进程互相看到的写无序，但最终一致。不对跨进程的消息排序。在课程“[🔗复制（三）：最早的数据复制方式竟然是无主复制？](#)”中讨论的 **Quorum** 机制就是最终一致性。


思考题

通过对一致性模型的学习，你可以通过读写操作序列，分别举出线性一致性、顺序一致性、因果一致性和最终一致性的例子吗？

欢迎你在留言区发表你的看法。如果这节课对你有帮助，也推荐你分享给更多的同事、朋友。

分享给需要的人，Ta订阅超级会员，你最高得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 25 | 事务（四）：持久性，吃一碗粉就付一碗粉的钱

下一篇 27 | 一致性与共识（二）：它们是鸡生蛋还是蛋生鸡？

精选留言 (1)

 写留言



peter

2022-04-01

请教老师一个问题啊：

Q1：顺序一致性的描述是否有矛盾？

“顺序一致性，它是指所有进程看到的事件历史一致有序，但不需要符合时间先后顺序，”，这句话中，竟然“历史一致”，“历史”就是表示时间啊，历史一致，肯定时间一致啊；但后面说“不需要时间符合先后顺序”。前后不矛盾吗？

作者回复: 这里的历史是以进程看到的为准，如果两个没有因果的事件 a,b，a 发生后，b 才发生，但是如果所有的进程都认为 b 先发生，那么事件历史就是 b,a。

