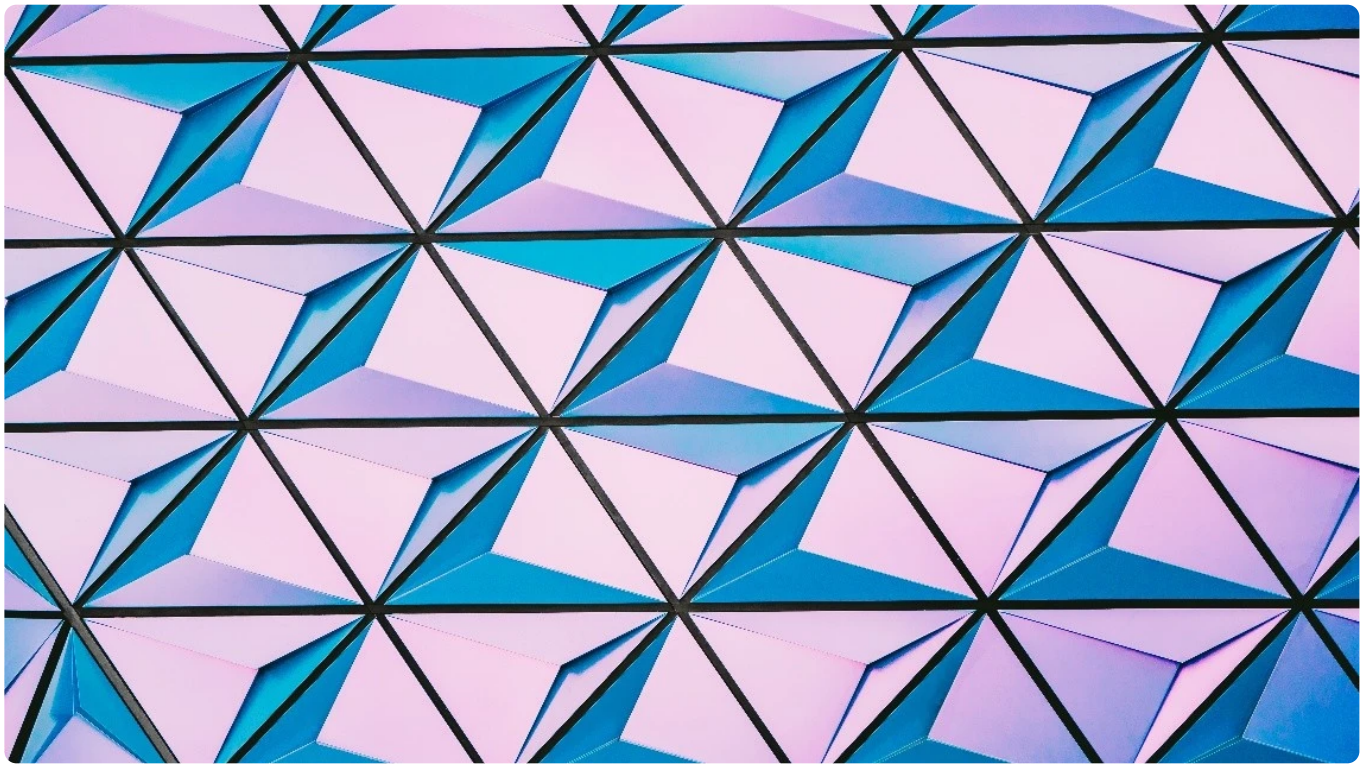
复习课（五）| Chubby

2021-11-24 黄金

《大数据经典论文解读》 课程介绍 >



讲述：王惠
时长 14:20 大小 13.13M

你好，我是黄金。今天我们来复习 Chubby 这篇论文。

Chubby 介绍

Chubby 是一种粗粒度的分布式锁服务，允许多个客户端通过分布式锁，就某项提议达成共识。它的**首要设计目标是可靠性、可用性和易用性**，而吞吐量和存储能力是次要的。

Chubby 的客户端接口类似 Unix 文件系统，可以方便地读写文件，在此之上它还提供了锁操作，以及诸如文件内容变化之类的事件变更通知机制。合理地使用 Chubby，我们[illegible]可以让它同时服务几万个客户端。

在 GFS 中，Chubby 用于选举 Master；在 Bigtable 中，Chubby 不仅用于选举 Master，还能够让 Master 方便地发现 Tablet Server 服务，也能让客户端轻松地找到 Master 地址。

课程内容梳理

徐老师通过 3 节课给我们讲透了 Chubby 论文，其内容的广度和深度远超论文本身。让我们先来回顾下课程内容。

第 1 讲首先提出了两个问题：GFS 如何保证集群中只有一个 Master？以及 Master 怎么把数据同步复制到 Backup Master，由此引出了**分布式共识问题**。接着讨论了分布式系统中同步复制数据的解决方案，也就是两阶段提交和三阶段提交，而不管是哪一种方式，我们都面临着协调者单点故障的问题。如果协调者需要自动选举，如何保证系统中只有一个协调者，问题又再次指向了分布式共识。

第 2 讲还是从数据库事务出发，先谈了 ACID 的概念，并重点聊了 ACID 中的 I，也就是不同的隔离级别。最严格的隔离级别是**可串行化**，可串行化要求事务在外部看来是一个一个顺序执行的，这样事务的隔离程度是上去了，但是数据库的性能也下来了，所以多数数据库产品在实现可串行化语义的时候，选择了让相关的事务顺序执行，不相关的事务并发执行。

而分布式共识和可串行化相同的地方在于，要保证操作在集群中的不同节点发生的顺序是一致的。讲到这里，也就到了第 2 讲的最后，分布式共识算法 Paxos 浮出水面。

第 3 讲才是正式介绍 Chubby。第一部分老师先回应了第 1 讲中提到的两个问题，也就是 GFS 中的 Master 和两阶段提交中的协调者选举问题，Paxos 是如何解决这类共识问题的，以及 Chubby 对 Paxos 的粗粒度锁封装又是怎么回事。第二部分则介绍了 Chubby 的系统架构、对外接口，以及对分布式锁的优化。

接下来我会和你一起复习下两阶段提交、数据库事务的隔离级别、Paxos 和 Chubby 这四个知识点，并谈一谈我的认识，期待我们可以一起交流看法。

两阶段提交

当一个事务操作需要跨越多个分布式节点的时候，就需要引入一个称为**协调者**的组件，来统一调度所有分布式节点的执行逻辑，这些被调度的节点被称为**参与者**。

两阶段提交是把事务提交分成了两个阶段。

> 阶段一，由协调者发出一个准备请求，询问所有参与者是否能提交事务，参与者收到请求后，如果回应能提交事务，就需要做好提交事务的准备，也就是写 redo 日志，同时也要做好撤销事务的准备，也就是写 undo 日志。
>
> 阶段二，协调者确认所有参与者都能提交事务，就会再发出一个执行请求，参与者收到请求后，会根据 redo 日志应用状态变更。如果有参与者在阶段一回应不能提交事务，协调者就会通过阶段二的请求，要求所有参与者撤销事务，参与者在收到请求后，会根据 undo 日志回滚状态变更。

两阶段提交有**两个关键点**，一个是通过唯一的协调者来调度所有参与者的执行逻辑，二个是参与者需要先作出操作能否执行的承诺，一旦承诺了就不能反悔，将来必须能够执行这个操作。正是这两个关键点，保证了跨节点的原子性。

那么，我们利用两阶段提交，就能够实现数据同步复制。不过如果协调者或参与者出现故障，两阶段提交就无法工作了。**怎么选举出新的协调者或参与者，让故障自动恢复呢？**这就需要利用分布式共识算法。

在回顾分布式共识算法之前，我们先来复习一下数据库事务的隔离级别，因为其中可串行化的隔离级别和分布式共识算法要求的可线性化，是有不少相似之处的，通过两者的对比，我们就可以更清楚地认识分布式共识算法的目标。

事务隔离级别

事务是将多个读写操作捆绑在一起的逻辑操作单元。事务的隔离级别可以分成四种，读未提交、读已提交、可重复读和可串行化。我们可以通过不同的加锁方式，来理解不同事务隔离级别的含义。加锁的方式主要有三种，读锁、写锁和范围锁。读锁和写锁我们应该都比较清楚，范围锁指的是对于某个范围加排他锁，在这个范围内的数据不能被读取，也不能被写入。

读未提交可以理解成，对事务涉及到的数据只加写锁，并且一直持续到事务结束，但完全不加读锁。所以一个事务，可以读到另一个事务已修改但未提交的数据。

读已提交可以理解成，对事务涉及到的数据加的写锁，并且一直持续到事务结束，但加的读锁在查询操作完成后就马上会释放。这样，一个事务不会读到另一个事务未提交的数据。但是一个事务如果多次读取某个数据，别的事务又在修改这个数据，那么多次读取的结果，就会受其他事务修改的影响而发生变化。

可重复读可以理解成，对事务所涉及到的数据加读锁和写锁，并且一直持续到事务结束。这样一个事务多次读取某个数据，就能看到相同的结果。不过如果一个事务多次查询某个范围的结果集，还是有可能看到不同的结果。

可串行化可以理解成，对事务所有读、写的数据全都加上读锁、写锁和范围锁，并且一直持续到事务结束。

以上说明仅仅能帮助我们理解不同事务隔离级别的含义，但不是说它们的实现方式就是这样。想要更深入地了解事务，我们可以去阅读《数据密集型应用系统设计》第 7 章“事务”。

那么，可串行化和可线性化其实是有相似之处的，两者都在表达按顺序执行的意思。而不同之处在于，**可串行化是保证事务按顺序执行**，这里的事务由多个读写操作构成，**可线性化只保证单个操作按顺序执行**。我们把 GFS 的 Master 的数据，同步复制到 Backup Master，是同步复制一个个独立的操作，它并不涉及到事务，所以只需要可线性化的支持。而分布式共识算法实现了可串行化，它可以让所有节点建立共识，按一致的顺序执行操作。

Paxos 算法

而提到了分布式共识算法，那就一定绕不开 Paxos，它是各种分布式共识算法的核心。

那么我们要怎么理解 Paxos 算法呢？我觉得可以抓住两点：**如何保证一个提议被所有节点接受；如何保证一组提议会被所有节点按相同顺序接受**。

首先，在分布式环境下，没有唯一的决策节点，所有的决策都是从多数节点的一致意见中产生的。剩余的少数节点，可以从多数节点中学习未知的决策。

GFS 中的某个进程是不是 Master，不是由这个进程自己说了算，而是由其他进程说了算。当多数进程认为这个进程是 Master，那它就是 Master。而如果多数进程不能就这个进程是否是 Master，达成一致意见，就会再启动一轮投票，直到选出唯一的 Master 为止。

在 Paxos 算法中，发起投票的是提案者，负责决策的是接受者，一个进程可以同时是提案者和接受者。Paxos 算法达成共识分成两步：

> 第一步是 **Prepare 阶段**，提案者发起提议，说接下来咱们执行操作 1 吧，所有的接受者都要确定要不要接受这个提议，如果不接受，需要告诉提案者，由别的提案者发起，并且已经被选中的最新提议是什么。
>
> 第二步是 **Accept 阶段**，如果提案者发起的提议被多数接受者接受了，提案者就可以再次广播这个提议，如果接受者发现这个提议是自己之前接受的最新提议，会再次确认接受。如果多数接受者都再次确认接受，那么这个提议就算真正被选中。

我在了解 Paxos 算法之后，第一个疑问就是，**为什么一次决策要经过两轮提议？如果只是需要多数接受者同意，一轮提议就行了。但是，需要两轮提议的原因，实际上是存在多个提案者，它们可能同时发起提议，任何一个提案者都需要知道最新的提案是哪个，以便确定自己发起的是一个更新的提案。新与旧的秘密就藏在提案编号**里，最大的编号就是最新的提案。

所以，第一轮提议的作用，是要让提案者获得最新的提案编号，第二轮提议才是提交自己的提案。这样就既保证了提议的选中，又保证了提议的顺序。

如果系统中只有一个提案者，那提案者肯定知道最新的提案编号是什么，于是两轮提议就可以简化为一轮提议。这也是 Multi Paxos 的优化思路。

Chubby 的架构设计

那么，Chubby 其实就是在 Paxos 之上，封装的粗粒度分布式锁服务。**分布式锁和粗粒度是 Chubby 的两个重要的设计决策。**

Chubby 没有设计成 Paxos 算法协议库，而是**设计成了需要访问中心化节点的分布式锁服务**，是因为分布式锁服务有四个优点。

第一个优点，是它对上层应用程序的侵入性更小。在系统开发初期，开发人员往往不会为系统的高可用性做出充分的考虑，随着系统的用户越来越多，高可用性也变得越来越重要，副本复制和主节点选举会被加入已有的设计中。相比共识算法的客户端库，分布式锁的侵入性更小，我们更容易在原来的系统中实现提供高可用性的需求。

第二个优点，是分布式锁服务便于提供数据的发布和订阅。服务在选举 Master 节点之后，往往需要把结果广播给其他节点。分布式锁服务就非常适合提供这个功能，这样也能减少上层应用程序对其他服务的依赖。

第三个优点，是开发人员对基于锁的接口更加熟悉。这种表面上的熟悉，就让说服开发人员使用 Chubby 这件事，变得容易很多。

第四个优点，是我们能更便捷地构建更可靠的服务。共识算法需要多个节点才能作出决策，使用共识算法的客户端库，应用程序需要自己管理多个节点，而使用分布式锁服务，应用程序只需要一个节点，就能使用它提供的高可靠功能。

而**粗粒度，是指外部客户端持有锁的时间比较长**，这个时间可能是数小时甚至数天。这种设计降低了 Chubby 服务的负载，使得 Chubby 可以同时服务更多的客户端。粗粒度的锁也有助于客户端在 Chubby 服务出现短暂失效的时候保持锁，这符合应用程序的期望。就拿 Master 选举来讲，它总是希望 Master 出现故障的时候重新选举，而不是 Chubby 出现抖动的时候重新选举。

另外，Chubby 的系统结构主要是由客户端库和服务端两部分组成的，外部客户端会通过客户端库提供的 RPC 调用与服务端进行通信。

Chubby 的服务端是一个集群，一般由 5 个节点构成，它们之中有一个节点会被选举成 Master，作为唯一的提案者。Master 的生命周期被称为**租期**，一个运行良好的 Master 会通过续租，不断延长自己的租期，而出现故障的 Master 会在租期结束后，被其他的节点代替。

我们可以把 Chubby 的服务端看成一个**三层的系统**。最底层，是一个 Paxos 协议实现的同步日志复制的系统；中间层，是一个简单的 KV 数据库；最上层，才是 Chubby 封装的锁服务。

小结

好了，到这里 Chubby 论文的内容我们就复习完了。

Chubby 论文本身并不难懂，难的是它背后的分布式问题，以及分布式共识。在学完了解读 Chubby 论文的这三节课之后，我也去阅读了老师推荐的《数据密集型应用系统设计》的第 7、8、9 章，对我启发也很大，所以如果你也想要深入了解 Chubby 以及与分布式相关的问题，建议你一定要去读一读。

上一篇 复习课（四）| Thrift

下一篇 复习课（六）| Hive

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。