06 | Paxos算法（二）：Multi-Paxos不是一个算法，而是统称

2020-02-24 韩健

讲述：于航
时长 10:08 大小 8.13M

你好，我是韩健。

经过上节课的学习，你应该知道，Basic Paxos 只能就单个值（Value）达成共识，一旦遇到为一系列的值实现共识的时候，它就不管用了。虽然兰伯特提到可以通过多次执行 Basic Paxos 实例（比如每接收到一个值时，就执行一次 Basic Paxos 算法）实现一系列值的共识。但是，很多同学读完论文后，应该还是两眼摸黑，虽然每个英文单词都能读懂，但还是不理解兰伯特提到的 Multi-Paxos，为什么 Multi-Paxos 这么难理解呢？

在我看来，兰伯特并没有把 Multi-Paxos 讲清楚，只是介绍了大概的思想，缺少算法过程的细节和编程所必须的细节（比如缺少选举领导者的细节）。这也就导致每个人实现的 Multi-Paxos 都不一样。不过从本质上看，大家都是在兰伯特提到的 Multi-Paxos 思想上

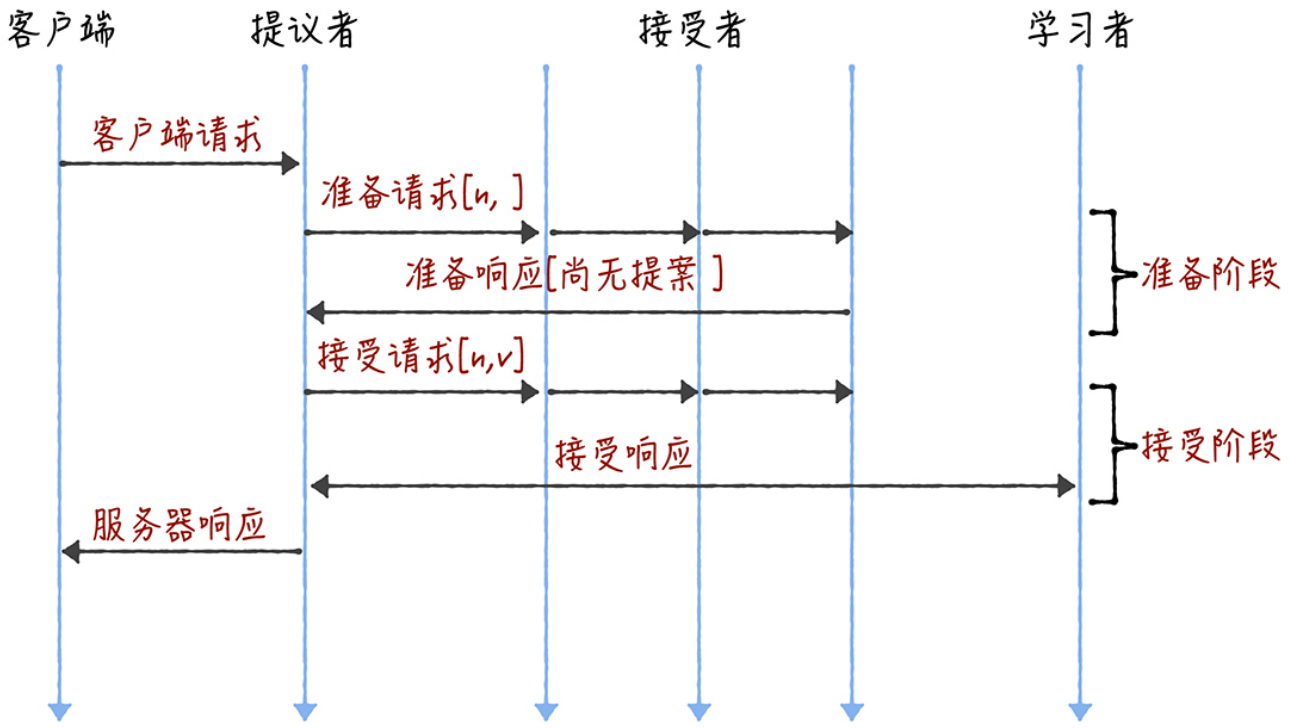
补充细节，设计自己的 Multi-Paxos 算法，然后实现它（比如 Chubby 的 Multi-Paxos 实现、Raft 算法、ZAB 协议等）。

所以在这里，我补充一下：**兰伯特提到的 Multi-Paxos 是一种思想，不是算法。而 Multi-Paxos 算法是一个统称，它是指基于 Multi-Paxos 思想，通过多个 Basic Paxos 实例实现一系列值的共识的算法（比如 Chubby 的 Multi-Paxos 实现、Raft 算法等）。**这一点尤其需要你注意。

为了帮你掌握 Multi-Paxos 思想，我会先带你了解，对于 Multi-Paxos 兰伯特是如何思考的，也就是说，如何解决 Basic Paxos 的痛点问题；然后我再以 Chubby 的 Multi-Paxos 实现为例，具体讲解一下。为啥选它呢？因为 Chubby 的 Multi-Paxos 实现，代表了 Multi-Paxos 思想在生产环境中的真正落地，它将一种思想变成了代码实现。

兰伯特关于 Multi-Paxos 的思考

熟悉 Basic Paxos 的同学（可以回顾一下05 讲）可能还记得，Basic Paxos 是通过二阶段提交来达成共识的。在第一阶段，也就是准备阶段，接收到大多数准备响应的提议者，才能发起接受请求进入第二阶段（也就是接受阶段）：



而如果我们直接通过多次执行 Basic Paxos 实例，来实现一系列值的共识，就会存在这样几个问题：

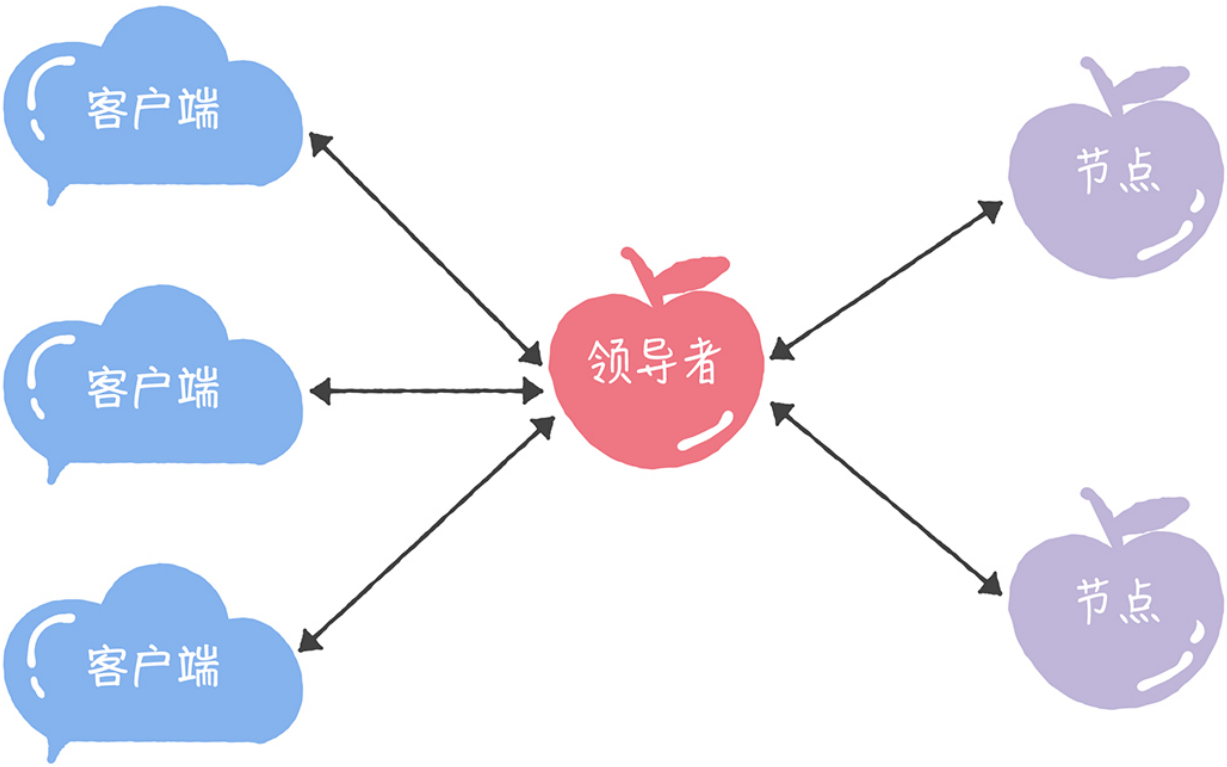
如果多个提议者同时提交提案，可能出现因为提案冲突，在准备阶段没有提议者接收到大多数准备响应，协商失败，需要重新协商。你想象一下，一个 5 节点的集群，如果 3 个节点作为提议者同时提案，就可能发生因为没有提议者接收大多数响应（比如 1 个提议者接收到 1 个准备响应，另外 2 个提议者分别接收到 2 个准备响应）而准备失败，需要重新协商。

2 轮 RPC 通讯（准备阶段和接受阶段）往返消息多、耗性能、延迟大。你要知道，分布式系统的运行是建立在 RPC 通讯的基础之上的，因此，延迟一直是分布式系统的痛点，是需要我们在开发分布式系统时认真考虑和优化的。

那么如何解决上面的 2 个问题呢？可以通过引入领导者和优化 Basic Paxos 执行来解决，咱们首先聊一聊领导者。

领导者（Leader）

我们可以通过引入领导者节点，也就是说，领导者节点作为唯一提议者，这样就不存在多个提议者同时提交提案的情况，也就不存在提案冲突的情况了：

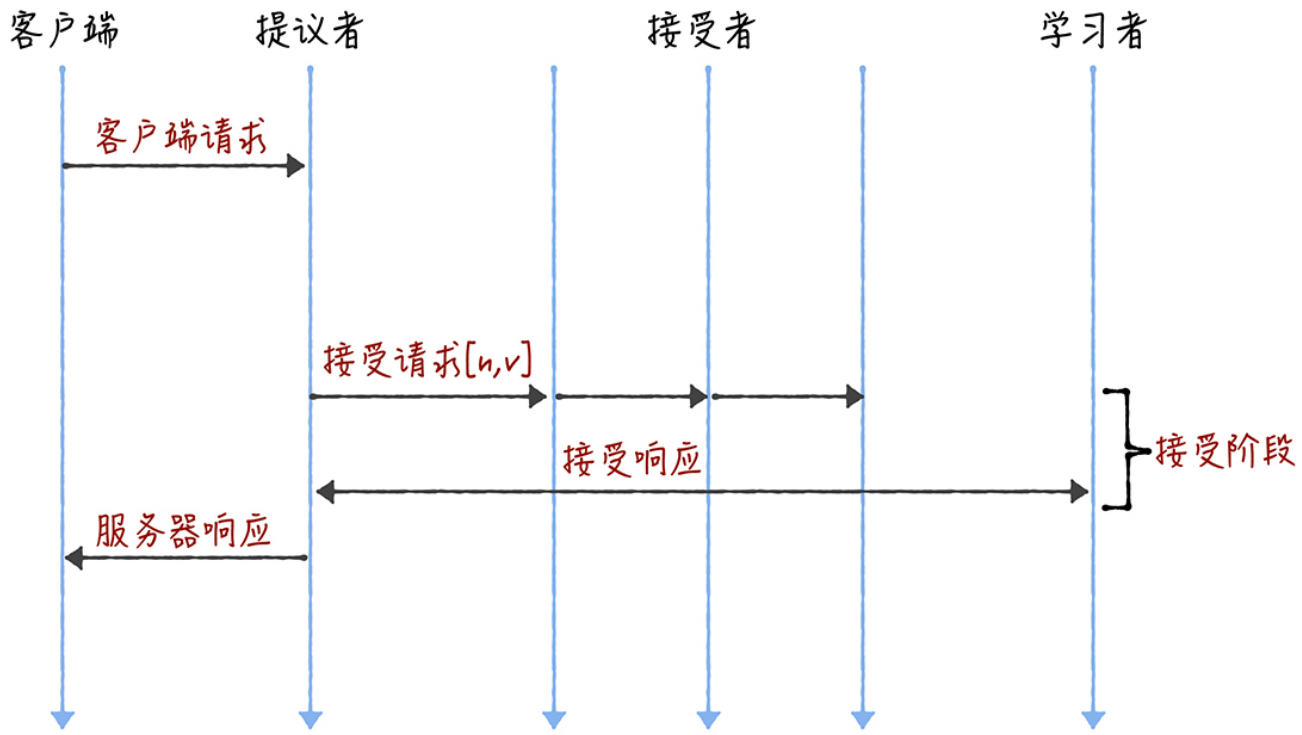


在这里，我补充一点：**在论文中，兰伯特没有说如何选举领导者，需要我们在实现 Multi-Paxos 算法的时候自己实现。** 比如在 Chubby 中，主节点（也就是领导者节点）是通过执行 Basic Paxos 算法，进行投票选举产生的。

那么，如何解决第二个问题，也就是如何优化 Basic Paxos 执行呢？

优化 Basic Paxos 执行

我们可以采用“当领导者处于稳定状态时，省掉准备阶段，直接进入接受阶段”这个优化机制，优化 Basic Paxos 执行。也就是说，领导者节点上，序列中的命令是最新的，不再需要通过准备请求来发现之前被大多数节点通过的提案，领导者可以独立指定提案中的值。这时，领导者在提交命令时，可以省掉准备阶段，直接进入到接受阶段：



你看，和重复执行 Basic Paxos 相比，Multi-Paxos 引入领导者节点之后，因为只有领导者节点一个提议者，只有它说了算，所以就不存在提案冲突。另外，当主节点处于稳定状态时，就省掉准备阶段，直接进入接受阶段，所以在很大程度上减少了往返的消息数，提升了性能，降低了延迟。

讲到这儿，你可能会问了：在实际系统中，该如何实现 Multi-Paxos 呢？接下来，我以 Chubby 的 Multi-Paxos 实现为例，具体讲解一下。

Chubby 的 Multi-Paxos 实现

既然兰伯特只是大概的介绍了 Multi-Paxos 思想，那么 Chubby 是如何补充细节，实现 Multi-Paxos 算法的呢？

首先，它通过引入主节点，实现了兰伯特提到的领导者（Leader）节点的特性。也就是说，主节点作为唯一提议者，这样就不存在多个提议者同时提交提案的情况，也就不存在提案冲突的情况了。

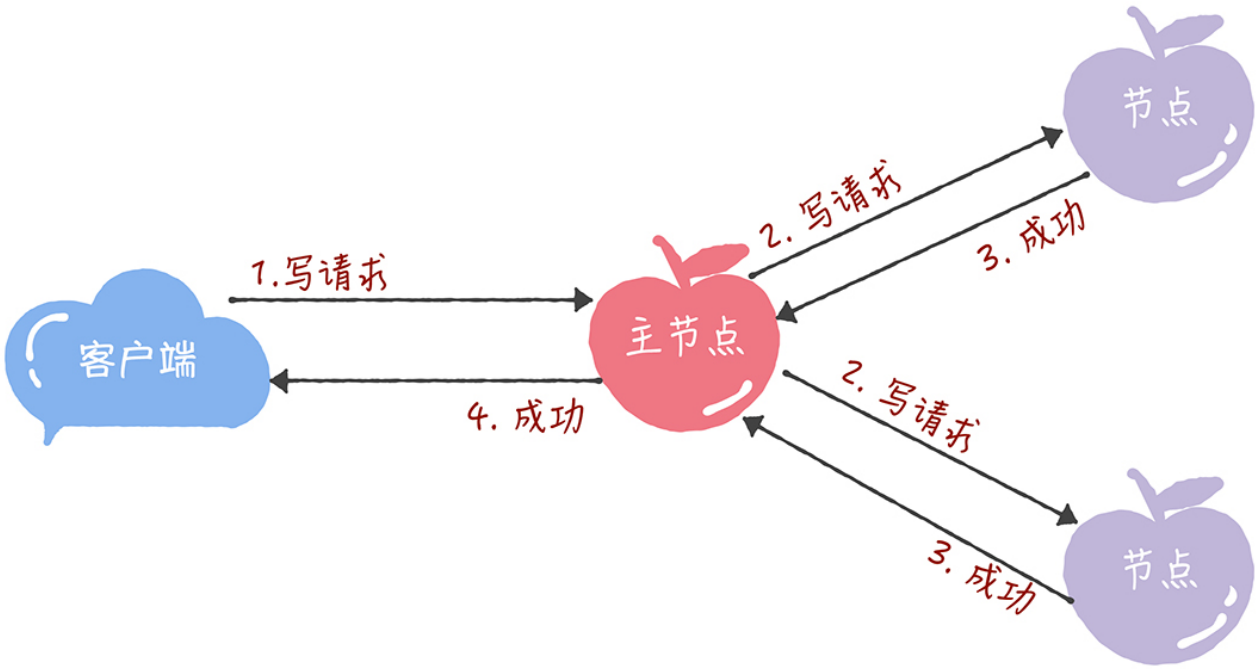
另外，在 Chubby 中，主节点是通过执行 Basic Paxos 算法，进行投票选举产生的，并且在运行过程中，主节点会通过不断续租的方式来延长租期（Lease）。比如在实际场景中，几天内都是同一个节点作为主节点。如果主节点故障了，那么其他的节点又会投票选举出新的主节点，也就是说主节点是一直存在的，而且是唯一的。

其次，在 Chubby 中实现了兰伯特提到的，“当领导者处于稳定状态时，省掉准备阶段，直接进入接受阶段”这个优化机制。

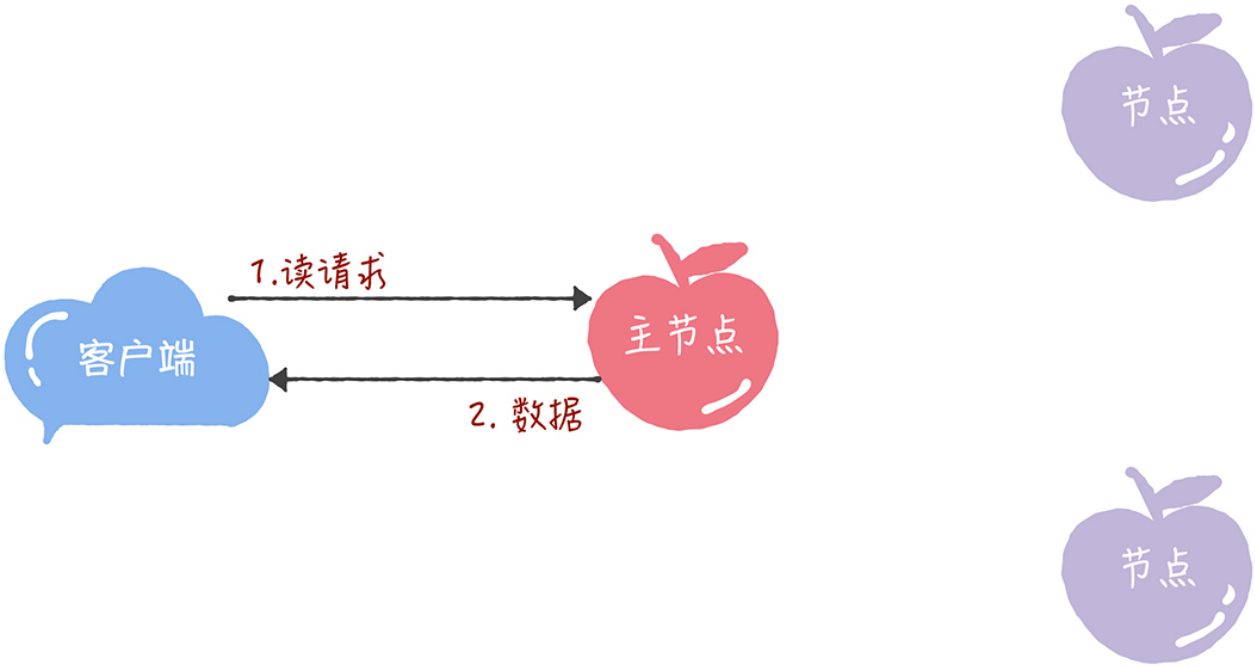
最后，在 Chubby 中，实现了成员变更（Group membership），以此保证节点变更的时候集群的平稳运行。

最后，我想补充一点：**在 Chubby 中，为了实现了强一致性，读操作也只能在主节点上执行。** 也就是说，只要数据写入成功，之后所有的客户端读到的数据都是一致的。具体的过程，就是下面的样子。

所有的读请求和写请求都由主节点来处理。当主节点从客户端接收到写请求后，作为提议者，执行 Basic Paxos 实例，将数据发送给所有的节点，并且在大多数的服务器接受了这个写请求之后，再响应给客户端成功：



当主节点接收到读请求后，处理就比较简单了，主节点只需要查询本地数据，然后返回给客户端就可以了：



Chubby 的 Multi-Paxos 实现，尽管是一个闭源的实现，但这是 Multi-Paxos 思想在实际场景中的真正落地，Chubby 团队不仅编程实现了理论，还探索了如何补充细节。其中的思考和设计非常具有参考价值，不仅能帮助我们理解 Multi-Paxos 思想，还能帮助我们理解其他的 Multi-Paxos 算法（比如 Raft 算法）。

内容小结

本节课我主要带你了解了 Basic Paxos 的局限，以及 Chubby 的 Multi-Paxos 实现。我希望你明确的重点如下：

1. 兰伯特提到的 Multi-Paxos 是一种思想，不是算法，而且还缺少算法过程的细节和编程所必须的细节，比如如何选举领导者等，这也就导致了每个人实现的 Multi-Paxos 都不一样。而 Multi-Paxos 算法是一个统称，它是指基于 Multi-Paxos 思想，通过多个 Basic Paxos 实例实现一系列数据的共识的算法（比如 Chubby 的 Multi-Paxos 实现、Raft 算法等）。
2. Chubby 实现了主节点（也就是兰伯特提到的领导者），也实现了兰伯特提到的 **“当领导者处于稳定状态时，省掉准备阶段，直接进入接受阶段”** 这个优化机制，省掉 Basic Paxos 的准备阶段，提升了数据的提交效率，但是所有写请求都在主节点处理，限制了集群处理写请求的并发能力，约等于单机。

3. 因为在 Chubby 的 Multi-Paxos 实现中，也约定了“大多数原则”，也就是说，只要大多数节点正常运行时，集群就能正常工作，所以 Chubby 能容错（n - 1）/2 个节点的故障。
4. 本质上而言，“当领导者处于稳定状态时，省掉准备阶段，直接进入接受阶段”这个优化机制，是通过减少非必须的协商步骤来提升性能的。这种方法非常常用，也很有效。比如，Google 设计的 QUIC 协议，是通过减少 TCP、TLS 的协商步骤，优化 HTTPS 性能。**我希望你能掌握这种性能优化思路，后续在需要时，可以通过减少非必须的步骤，优化系统性能。**

最后，我想说的是，我个人比较喜欢 Paxos 算法（兰伯特的 Basic Paxos 和 Multi-Paxos），虽然 Multi-Paxos 缺失算法细节，但这反而给我们提供了思考空间，让我们可以反复思考和考据缺失的细节，比如在 Multi-Paxos 中到底需不需要选举领导者，再比如如何实现提案编号等等。

但我想强调，Basic Paxos 是经过证明的，而 Multi-Paxos 是一种思想，缺失实现算法的必须编程细节，这就导致，Multi-Paxos 的最终算法实现，是建立在一个未经证明的基础之上的，正确性是个问号。

与此同时，实现 Multi-Paxos 算法，最大的挑战是如何证明它是正确的。 比如 Chubby 的作者做了大量的测试，和运行一致性检测脚本，验证和观察系统的健壮性。在实际使用时，我不推荐你设计和实现新的 Multi-Paxos 算法，而是建议优先考虑 Raft 算法，因为 Raft 的正确性是经过证明的。当 Raft 算法不能满足需求时，你再考虑实现和优化 Multi-Paxos 算法。

课堂思考

既然，我提了 Chubby 只能在主节点上执行读操作，那么在最后，我给你留了一个思考题，这个设计有什么局限呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

上一篇　05 | Paxos算法（一）：如何在多个节点间确定某变量的值?

精选留言 (15)

写留言

HuaMax 置顶
2020-02-24

“当领导者处于稳定状态时，省掉准备阶段，直接进入接受阶段”这个优化机制。请问，什么样是稳定状态？为什么稳定状态可以省掉准备阶段？

作者回复: 如何理解领导者处于稳定状态?
领导者节点上，序列中的命令是最新的，不再需要通过准备请求来发现之前被大多数节点通过的提案，领导者可以独立指定提案中的值。
我来具体说说，准备阶段的意义，是发现接受者节点上，已经通过的提案的值。如果在所有接受者节点上，都没有已经通过的提案了，这时，领导者就可以自己指定提案的值了，那么，准备阶段就没有意义了，也就是可以省掉了。

1

每天晒白牙 置顶
2020-02-24

只能在主节点进行读操作，效果相当于单机，对吞吐量和性能有所影响
写也是在主节点进行，性能也有问题

展开

作者回复: 加一颗星:)

约书亚
2020-02-24

本来想问个问题，看到思考题提到了。
从raft和zab的实现来看，一致性读操作的处理和写操作是类似的，不从本地读，而是也要发请求到所有节点，得到大多数节点的响应才行。我了解到的有的实现是领导者发送一个空操作给所有节点。
这样做的原因不光是考虑吞吐量的问题，而是读本地是满足不了强一致性的，因为自以...

展开

作者回复: 领导者节点上，序列中的命令是最新的，不再需要通过准备请求来发现之前被大多数节点通过的提案，领导者可以独立指定提案中的值。
我来具体说说，准备阶段的意义，是发现接受者节点上，已经通过的提案的值。如果在所有接受者节点上，都没有已经通过的提案了，这时，领导者就可以自己指定提案的值了，那么，准备阶段就没有意义了，也就是可以省掉了。

1

Dovelol
2020-02-24

老师好，想问下，如果只有领导者可以发起提案，那么这是不是就退化为串行操作了，这样的话性能怎么保证呢，实际应用中是怎么解决的？我觉得最后问题， Chubby只能在主节点上执行读操作，在读请求量非常大的情况下，也是会遇到瓶颈的，还有就是单点问题，主节点挂了，在选出主节点之前就不能提供服务了对吧，该如果解决这类问题呢？

展开

2 1

Geek_MYMSwen
2020-02-24

Chubby的局限可能在于高读的系统中，如果读请求过大，会导致系统的不可用。另外在系统中如何能够将主节点更替的信息向用户传播也是需要考虑的问题。
还有有一种情况我没有想清楚，请各位指点：

一个分布式系统中有5个节点，3个在一个机房A(机器编号A1，A2，A3)，2个在另一个机房B（机器编号B1，B2）。 1）如果节点A1的机架网络发生故障，导致A1与其他节点通...

展开

1 1

zjm_tmac
2020-02-25

leader怎么确定acceptor的总数呢？集群是允许扩容的吗

作者回复: 怎么确定acceptor总数，涉及代码实现和成员变更；扩容涉及到成员变更。
首先，使用什么数据结构来保存acceptor总数，这属于编程实现，不属于算法了，绝大多数算法都不会约定的这么细的。
其次，Multi-Paxos，只是一种思想，缺少算法细节和编程所必须的细节，比如，成员变更，在Multi-Paxos中，提了下，可以把服务器配置作为指令，通过状态机提交，等等。但是，如果学习了09讲后，你就会发现，真正实现起来，比这个要复杂很多，比如Raft设计了2种成员变更方法，一种难以实现，一种容易实现，但在16年时，爆出了一个算法bug，虽然很快就修复了，但也反映了成员变更比较复杂，不是三言两语能讲清楚的。
另外，其实，学习Multi-Paxos的最好的方式，是先理解Raft，再回过头来，学习Multi-Paxos。如果在学习Multi-Paxos中遇到不理解的，可以在学习完Raft后，再回头来研究学习。

zjm_tmac
2020-02-25

看了下微信的PhxPaxos实现文章，确定多个值是通过多组paxos实例完成的，这篇文章好像没提高，到底是多组实例还是一组实例呢?

展开

作者回复: 多个值，更确切的说，是一系列值，是需要多次执行Basic Paxos实例的，文中也反复提到了哈。

小晏子
2020-02-24

Chubby 只能在主节点上执行读写操作, 那么这个主节点就是热点，所有的请求都要经过它，显而易见它就是系统的瓶颈，影响系统的并发度，而且在并发时，写请求会block读请求，影响整个系统的QPS。

作者回复: 加一颗星:)

Eric
2020-02-24

老师你好!
1. 领导者处于稳定状态是指当前只有一个领导者吗?
2. BasicPaxos只能就一个值达成一致 那么能详细讲讲MultiPaxos是怎么让一系列值打成一致的吗?

sai
2020-02-24

您好，假设有3台节点 A, B, C. leader 最开始是A, 依次执行写入操作[set x=1, set y=2, set z=3], 假设B和C都有可能超时，根据paxos只需要大多数写入成功就算执行成功的原则，当前状态可能为A:[x:1, y:2, z:3], B:[x:1, z:3], C: [y2, z3]。如果这个时候主节点A宕机，如何重新选择主节点并恢复数据呢?

展开

sword@zh
2020-02-24

“如果多个提议者同时提交提案，可能出现因为提案冲突，在准备阶段没有提议者接收到大多数准备响应，协商失败，需要重新协商。”

根据上文的介绍，接受者会接受提案编号最大的提案吧?

展开

1

Purson
2020-02-24

局限在于读写都是在主节点进行，性能相当于单机。假设主节点down机，需要重新选取主节点，此时如果发生大量读写请求，性能问题突显，可能会导致系统不可用，写请求可能丢失。

Jialin
2020-02-24

Chubby 只能在主节点上执行读操作，这个设计导致高并发情况下读操作的吞吐量受到限制，影响系统的可用性，但是保证了读数据一致性问题。

展开

忆水寒

2020-02-24

只能在主节点上执行读操作,有什么缺陷呢?
这样就相当于单机CA了，在大量读取操作时候，可能会使leader挂掉，导致服务不可用。要解决这个问题，就得写的时候由leader进行提交。读的时候由所有节点都可以响应。

展开

每天晒白牙

2020-02-24

如果直接通过多次执行 Basic Paxos 实例来达到共识有两个问题
1.如果多个提议者同时提交提案，可能出现因为提案冲突，在准备阶段没有提议者收到大多数准备响应，协商失败，这样就需要重新协商
2.因为准备阶段和接受阶段会进行两轮RPC通讯，往返消息多，耗性能，延迟大，这是需要优化的...

展开