浅谈分布式系统的基本问题：可用性与一致性

[lijinhui](https://yq.aliyun.com/users/43316) 2016-01-11 14:05:17 浏览6067 评论0

[分布式系统与计算](https://yq.aliyun.com/tags/type_blog-tagid_39/) [阿里技术协会](https://yq.aliyun.com/tags/type_blog-tagid_523/) [cap](https://yq.aliyun.com/tags/type_blog-tagid_771/) [paxos](https://yq.aliyun.com/tags/type_blog-tagid_772/) [zab](https://yq.aliyun.com/tags/type_blog-tagid_773/)

**摘要：** 该文章来自于阿里巴巴技术协会（ATA）精选文章。 背景         可用性（Availability）和一致性（Consistency）是分布式系统的基本问题，先有著名的CAP理论定义过分布式环境下二者不可兼得的关系，又有神秘的Paxos协议号称是史上最简单的分布式系统一致性算法并获得图灵奖，再有开源产品ZooKeeper实现的ZAB协议号称超越Paxos，它们之间究竟有什么联系？在网络上没有文章将其清楚地阐述过，于是想到把自己对CAP理论、Paxos协议以及ZAB协议的理解整理成短文，但我唯一不保证的是正确性，各位看官看着办。

该文章来自于阿里巴巴技术协会（ATA）精选文章。

背景

        可用性（Availability）和一致性（Consistency）是分布式系统的基本问题，先有著名的CAP理论定义过分布式环境下二者不可兼得的关系，又有神秘的Paxos协议号称是史上最简单的分布式系统一致性算法并获得图灵奖，再有开源产品ZooKeeper实现的ZAB协议号称超越Paxos，它们之间究竟有什么联系？在网络上没有文章将其清楚地阐述过，于是想到把自己对CAP理论、Paxos协议以及ZAB协议的理解整理成短文，但我唯一不保证的是正确性，各位看官看着办。

分布式系统的挑战

         一致性可理解为所有节点都能访问到最新版本的数据，这在单机场景下非常容易实现，使用共享内存和锁即可解决，但数据存储在单机会有两个限制：1）单机不可用系统整体将不可用；2）系统吞吐量受限于单机的计算能力。消除这两个限制的方法是用多机来存储数据的多个副本，负责更新的客户端会同时更新数据的多个副本，于是问题就来了，多机之间的网络可能无法连接，当负责更新的客户端无法同时到连接多个机器时，如何能保证所有客户端都能读到最新版本的数据？

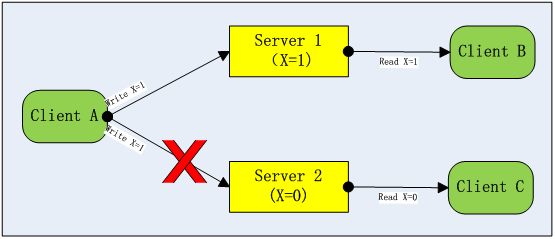
         如下图1中所示，Client A负责更新数据，为了保证Server 1和Server 2上的数据是一致的，Client A会将X=1的写操作同时发给Server 1和Server 2，但是当Client A和Server 2之间发生网络分区（网络无法连接）时，此时如果让write X=1的写操作在Server 1上成功，那Client B和Client C将从Server 1和Server 2上读取到不一致的X值；此时如果要保持X值的一致性，那么write X=1的写操作在Server 1和Server 2上都必须失败，这就是著名的CAP理论：在容忍网络分区的前提下，要么牺牲数据的一致性，要么牺牲写操作的可用性。  


图1：CAP理论示意图

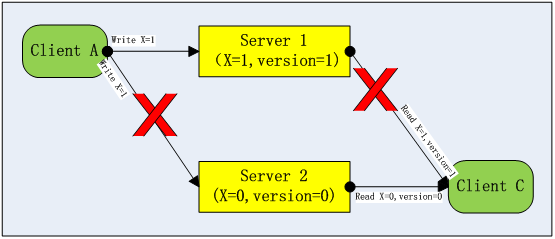
        解决这个问题你可能会想到让Client C同时读取Server 1和Server 2上的X值和版本信息，然后取Server 1和Server 2最新版本的X值, 如下图2所示。但Client C和Server 1之间也可能发生网络分区，这本质上是牺牲读可用性换取写可用性，并没有突破CAP理论。  


图2：对图1中可用性的优化

CAP理论

         CAP理论由加州大学伯克利分校的计算机教授Eric Brewer在2000年提出，其核心思想是任何基于网络的数据共享系统最多只能满足数据一致性(Consistency)、可用性(Availability)和网络分区容忍(Partition Tolerance)三个特性中的两个，三个特性的定义如下：

* 数据一致性：等同于所有节点拥有数据的最新版本
* 可用性：数据具备高可用性
* 分区容忍：容忍网络出现分区，分区之间网络不可达

        在大规模的分布式环境下，网络分区是必须容忍的现实，于是只能在可用性和一致性两者间做出选择，CAP理论似乎给分布式系统定义了一个悲观的结局，一时间大家都按照CAP理论在对热门的分布式系统进行判定，譬如认为HBase是一个CP系统，Cassandra是AP系统，我个人认为这是不严谨的，理由是CAP理论是对分布式系统中一个数据无法同时达到可用性和一致性的断言，而一个系统中往往存在很多类型的数据，部分数据（譬如银行账户中的余额）是需要强一致性的，而另外一部分数据（譬如银行的总客户数）并不要求强一致性，所以拿CAP理论来划分整个系统是不严谨的， CAP理论带来的价值是指引我们在设计分布式系统时需要区分各种数据的特点，并仔细考虑在小概率的网络分区发生时究竟为该数据选择可用性还是一致性。

        对CAP理论的另外一种误读是系统设计时选择其一而完全不去优化另外一项，可用性和一致性的取值范围并不是只有0和1，可用性的值域可以定义成0到100%的连续区间，而一致性也可分为强一致性、弱一致性、读写一致性、最终一致性等多个不同的强弱等级，细想下去CAP理论定义的其实是在容忍网络分区的条件下，“强一致性”和“极致可用性”无法同时达到。（注：这里用“极致可用性”而不用“100%可用性”是因为即使不考虑一致性，多台server组成的分布式系统也达不到100%的可用性，如果单个server的可用性是P，那n台server的极致可用性是http://img3.tbcdn.cn/L1/461/1/cc9c8bd3e008bf72a57f259ff14b79384dc1ae68, 公式的意思是只要任何一台或多台server可用就认为系统都是可用的）

        虽然无法达到同时达到强一致性和极致可用性，但我们可以根据数据类型在二者中选择其一后去优化另外一个，Paxos协议就是一种在保证强一致性前提下把可用性优化到极限的算法。

Paxos协议

        Paxos协议由Leslie Lamport最早在1990年提出，由于Paxos在云计算领域的广泛应用Leslie Lamport因此获得了2013年度图灵奖。

        Paxos协议提出只要系统中2f+1个节点中的f+1个节点可用，那么系统整体就可用并且能保证数据的强一致性，它对于可用性的提升是极大的，仍然假设单节点的可用性是P，那么2f+1个节点中任意组合的f+1以上个节点正常的可用性P(total)=http://img1.tbcdn.cn/L1/461/1/d8f33a62f83c661dc33b354779cdd03481d4a66e,又假设P=0.99，f=2，P(total)=0.9999901494，可用性将从单节点的2个9提升到了5个9，这意味着系统每年的宕机时间从87.6小时降到0.086小时，这已经可以满足地球上99.99999999%的应用需求。

        Leslie写的两篇论文：《The Part-Time Parliament》和《Paxos Made Simple》比较完整的阐述了Paxos的工作流程和证明过程，Paxos协议把每个数据写请求比喻成一次提案（proposal），每个提案都有一个独立的编号，提案会转发到提交者（Proposer）来提交，提案必须经过2f+1个节点中的f+1个节点接受才会生效，2f+1个节点叫做这次提案的投票委员会(Quorum)，投票委员会中的节点叫做Acceptor，Paxos协议流程还需要满足两个约束条件： a）Acceptor必须接受它收到的第一个提案；b）如果一个提案的v值被大多数Acceptor接受过，那后续的所有被接受的提案中也必须包含v值（v值可以理解为提案的内容，提案由一个或多个v和提案编号组成）。

        Paxos协议流程划分为两个阶段，第一阶段是Proposer学习提案最新状态的准备阶段；第二阶段是根据学习到的状态组成正确提案提交的阶段，完整的协议过程如下：

**阶段 1.**

1. Proposer选择一个提案编号*n* ，然后向半数以上的Acceptors发送编号为 *n* 的prepare请求。
2. 如果一个Acceptor收到一个编号为*n* 的prepare请求，且 *n* 大于它已经响应的所有prepare请求的编号，那么它就会保证不会再通过(accept)任何编号小于 *n* 的提案，同时将它已经通过的最大编号的提案(如果存在的话)作为响应。

**阶段 2.**

1. 如果Proposer收到来自半数以上的Acceptor对于它的prepare请求(编号为*n* )的响应，那么它就会发送一个针对编号为 *n* ，value值为 *v* 的提案的accept请求给Acceptors，在这里 *v* 是收到的响应中编号最大的提案的值，如果响应中不包含提案，那么它就是任意值。
2. 如果Acceptor收到一个针对编号*n* 的提案的accept请求，只要它还未对编号大于 *n* 的prepare请求作出响应，它就可以通过这个提案。

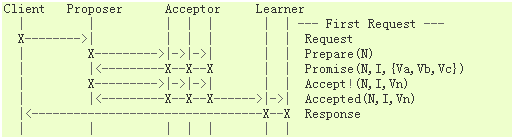
        用时序图来描述Paxos协议如图3所示：  


图3：Paxos协议流程的时序图

        上述Paxos协议流程看起来比较复杂，是因为要保证很多边界条件下的协议完备性，譬如初试值为空、两个Proposer同时提交提案等情况，但Paxos协议的核心可以简单描述为：Proposer先从大多数Acceptor那里学习提案的最新内容，然后根据学习到的编号最大的提案内容组成新的提案提交，如果提案获得大多数Acceptor的投票通过就意味着提案被通过。由于学习提案和通过提案的Acceptor集合都超过了半数，所以一定能学到最新通过的提案值，两次提案通过的Acceptor集合中也一定存在一个公共的Acceptor，在满足约束条件b时这个公共的Acceptor时保证了数据的一致性，于是Paxos协议又被称为多数派协议。

        Paxos协议的真正伟大之处在于它的简洁性，Paxos协议流程中任何消息都是可以丢失的，一致性保证并不依赖某个特殊消息传递的成功，这极大的简化了分布式系统的设计，极其匹配分布式环境下网络可能分区的特点，相比较在Paxos协议之前的“两阶段提交（2PC）”也能保证数据强一致性，但复杂度相当高且依赖单个协调者的可用性。

        那既然Paxos如此强大，那为什么还会出现ZAB协议？

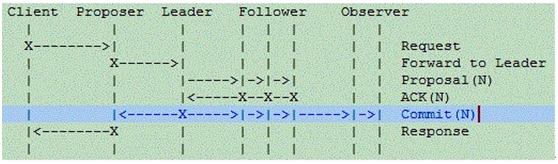
ZAB协议

        Paxos协议虽然是完备的，但要把它应用到实际的分布式系统中还有些问题要解决：

* 在多个Proposer的场景下，Paxos不保证先提交的提案先被接受，实际应用中要保证多提案被接受的先后顺序怎么办？
* Paxos允许多个Proposer提交提案，那有可能出现活锁问题，出现场景是这样的：提案n在第二阶段还没有完成时，新的提案n+1的第一阶段prepare请求到达Acceptor，按协议规定Acceptor将响应新提案的prepare请求并保证不会接受小于n+1的任何请求，这可能导致提案n将不会被通过，同样在n+1提案未完成第二阶段时，假如提案n的提交者又提交了n+2提案，这可能导致n+1提案也无法通过。
* Paxos协议规定提案的值v只要被大多数Acceptor接受过，后续的所有提案不能修改值v，那现实情况下我还要修改v值怎么办？

        ZooKeeper的核心算法ZAB通过一个简单的约束解决了前2个问题：所有提案都转发到唯一的Leader（通过Leader选举算法从Acceptor中选出来的）来提交，由Leader来保证多个提案之间的先后顺序，同时也避免了多Proposer引发的活锁问题。

        ZAB协议的过程用时序图描述如图4所示，相比Paxos协议省略了Prepare阶段，因为Leader本身就有提案的最新状态，不需要有提案内容学习的过程,图中的Follower对应Paxos协议中的Acceptor，Observer对应Paxos中的Learner。

  
图4：ZAB协议的工作过程

        ZAB引入Leader后也会带来一个新问题： Leader宕机了怎么办？其解决方案是选举出一个新的Leader，选举Leader的过程也是一个Paxos提案决议过程，这里不展开讨论。

        那如何做到提案的值v可以修改呢？这不是ZAB协议的范畴，研究ZooKeeper源码后发现它是这么做的：ZooKeeper提供了一个znode的概念，znode可以被修改，ZooKeeper对每个znode都记录了一个自增且连续的版本号，对znode的任何修改操作（create/set/setAcl）都会促发一次Paxos多数派投票过程，投票通过后znode版本号加1，这相当于用znode不同版本的多次Paxos协议来破除单次Paxos协议无法修改提案值的限制。

        从保证一致性的算法核心角度看ZAB确实是借鉴了Paxos的多数派思想，但它提供的全局时序保证以及ZooKeeper提供给用户可修改的znode才让Paxos在开源界大放异彩，所以ZAB的价值不仅仅是提供了Paxos算法的优化实现，也难怪ZAB的作者一直强调ZAB和Paxos是不一样的算法。

总结

         CAP理论告诉我们在分布式环境下网络分区无法避免，需要去权衡选择数据的一致性和可用性，Paxos协议提出了一种极其简单的算法在保障数据一致性时最大限度的优化了可用性，ZooKeeper的ZAB协议把Paxos更加简化，并提供全局时序保证，使得Paxos能够广泛应用到工业场景。