原 分布式理论之一：Paxos算法的通俗理解

分布式理论之一：Paxos算法的通俗理解

维基的简介：Paxos算法是莱斯利·兰伯特（Leslie Lamport，就是 LaTeX 中的"La"，此人现在在微软研究院）于1990年提出的一种基于消息传递且具有高度容错特性的一致性算法。

Paxos算法目前在Google的Chubby、MegaStore、Spanner等系统中得到了应用，Hadoop中的ZooKeeper也使用了Paxos算法，在上面的各个系统中，使用的算法与Lamport提出的原始Paxos并不完全一样，这个以后再慢慢分析。本博文的目的是，如何让一个小白在半个小时之内理解Paxos算法的思想。小白可能对数学不感兴趣，对分布式的复杂理论不熟悉，只是一个入门级程序员。之所以想写这篇博文，是因为自己看了网上很多介绍Paxos算法的文章，以及博客，包括Lamport的论文，感觉还是难以理解，大多过于复杂，本人一直认为，复杂高深的理论背后一定基于一些通用的规律，而这些通用的规律在生活中其实我们早就遇到过，甚至用过。所以，我们先忽略Paxos算法本身，从生活中的小事开始谈起。

假如有一群驴友要决定中秋节去旅游，这群驴友分布在全国各地，假定一共25个人，分别在不同的省，要决定到底去拉萨、昆明、三亚等等哪个地点（会合时间中秋节已经定了，此时需要决定旅游地）。最直接的方式当然就是建一个QQ群，大家都在里面投票，按照少数服从多数的原则。这种方式类似于“共享内存”实现的一致性，实现起来简单，但Paxos算法不是这种场景，因为Paxos算法认为这种方式有一个很大的问题，就是QQ服务器挂掉怎么办？Paxos的原则是容错性一定要很强。所以，Paxos的场景类似于这25个人相互之间只能发短信，需要解决的核心问题是，哪怕任意的一部分人（Paxos的目的其实是少于半数的人）“失联”了，其它人也能够在会合地点上达成一致。好了，怎么设计呢?

这25个人找了另外的5个人（当然这5个人可以从25个人中选，这里为了描述方便，就单拿出另外5个人），比如北京、上海、广州、深圳、成都的5个人，25个人都给他们发短信，告诉自己倾向的旅游地。这5个人相互之间可以并不通信，只接受25个人发过来的短信。这25个人我们称为驴友，那5个人称为队长。

先来看驴友的逻辑。驴友可以给任意5个队长都发短信，发短信的过程分为两个步骤：

第一步（申请阶段）：询问5个队长，试图与队长沟通旅游地。因为每个队长一直会收到不同驴友的短信，不能跟多个驴友一起沟通，在任何时刻只能跟一个驴友沟通，按照什么原则才能做到公平公正公开呢？这些短信都带有发送时间，队长采用的原则是同意与短信发送时间最新的驴友沟通，如果出现了更新的短信，则与短信更新的驴友沟通。总之，作为一个有话语权的人，只有时刻保持倾听最新的呼声，才能做出最明智的选择。在驴友发出短信后，等着队长回复。某些队长可能会回复说，你这条短信太老了，我不与你沟通；有些队长则可能返回说，你的短信是我收到的最新的，我同意跟你沟通。对于后面这些队长，还得返回自己决定的旅游地。关于队长是怎么决定旅游地的，后面再说。

对于驴友来说，第一步必须至少有半数以上队长都同意沟通了，才能进入下一步。否则，你连沟通的资格都没有，一直在那儿狂发吧。你发的短信更新，你获得沟通权的可能性才更大。。。。。。

因为至少有半数以上队长（也就是3个队长以上）同意，你才能与队长们进行实质性的沟通，也就是进入第二步；而队长在任何时候只能跟1个驴友沟通，所以，在任何时候，不可能出现两个驴友都达到了这个状态。。。当然，你可以通过狂发短信把沟通权抢了。。。。

第一种情况：跟A沟通的队长们（不一定是全部5个队长，但是半数以上）全部都还没有决定到底去那儿旅游，此时驴友A心花怒放，给这些队长发第二条短信，告诉他们自己希望的旅游地（比如马尔代夫）；

可能会收到两种结果：一是半数以上队长都同意了，于是表明A建议的马尔代夫被半数以上队长都同意了，整个决定过程完毕了，其它驴友迟早会知道这个消息的，A先去收拾东西准备去马尔代夫；除此之外，表明失败。可能队长出故障了，比如某个队长在跟女朋友打电话等等，也可能被其它驴友抢占沟通权了（因为队长喜新厌旧嘛，只有要更新的驴友给自己发短信，自己就与新人沟通，A的建议队长不同意）等等。不管怎么说，苦逼的A还得重新从第一步开始，重新给队长们发短信申请。

第二种情况：至少有一个队长已经决定旅游地了，A可能会收到来自不同队长决定的多个旅游地，这些旅游地是不同队长跟不同驴友在不同时间上做出的决定，那么，A会先看一下，是不是有的旅游地已经被半数以上队长同意了（比如3个队长都同意去三亚，1个同意去昆明，另外一个没搭理A），如果出现了这种情况，那就别扯了，说明整个决定过程已经达成一致了，收拾收拾准备去三亚吧，结束了；如果都没有达到半数以上（比如1个同意去昆明，1个同意去三亚，2个同意去拉萨，1个没搭理我），A作为一个高素质驴友，也不按照自己的意愿乱来了（Paxos的关键所在，后者认同前者，否则整个决定过程永无止境），虽然自己原来可能想去马尔代夫等等。就给队长们发第二条短信的时候，告诉他们自己希望的旅游地，就是自己收到的那堆旅游地中最新决定的那个。（比如，去昆明那个是北京那个队长前1分钟决定的，去三亚的决定是上海那个队长1个小时之前做出来的，于是顶昆明）。驴友A的想法是，既然有队长已经做决定了，那我就干脆顶最新那个决定。

从上面的逻辑可以看出，一旦某个时刻有半数以上队长同意了某个地点比如昆明，紧跟着后面的驴友B继续发短信时，如果获得沟通权，因为半数以上队长都同意与B沟通了，说明B收到了来自半数以上队长发过来的消息，B必然会收到至少一个队长给他发的昆明这个结果（否则说明半数以上队长都没有同意昆明这个结果，这显然与前面的假设矛盾），B于是会顶这个最新地点，不会更改，因为后面的驴友都会顶昆明，因此同意昆明的队长越来越多，最终必然达成一致。

看完了驴友的逻辑，那么队长的逻辑是什么呢?

队长的逻辑比较简单。

在申请阶段，队长只会选择与最新发申请短信的驴友沟通，队长知道自己接收到最新短信的时间，对于更老的短信，队长不会搭理；队长同意沟通了的话，会把自己决定的旅游地（或者还没决定这一信息）发给驴友。

在沟通阶段，驴友C会把自己希望的旅游地发过来（同时会附加上自己申请短信的时间，比如3分钟前），所以队长要检查一下，如果这个时间（3分钟前）确实是当前自己最新接收到申请短信的时间（说明这段时间没有驴友要跟自己沟通），那么，队长就同意驴友C的这个旅游地了（比如昆明，哪怕自己1个小时前已经做过去三亚的决定，谁让C更新呢，于是更新为昆明）；如果不是最新的，说明这3分钟内又有其它驴友D跟自己申请了，因为自己是个喜新厌旧的家伙，同意与D沟通了，所以驴友C的决定自己不会同意，等着D一会儿要发过来的决定吧。

Paxos的基本思想大致就是上面的过程。可以看出，其实驴友的策略才是Paxos的关键。让我们来跟理论对应一下。

Paxos主要用于保证分布式存储中副本（或者状态）的一致性。副本要保持一致，那么，所有副本的更新序列就要保持一致。因为数据的增删改查操作一般都存在多个客户端并发操作，到底哪个客户端先做，哪个客户端后做，这就是更新顺序。如果不是分布式，那么可以利用加锁的方法，谁先申请到锁，谁就先操作。但是在分布式条件下，存在多个副本，如果依赖申请锁+副本同步更新完毕再释放锁，那么需要有分配锁的这么一个节点（如果是多个锁分配节点，那么又出现分布式锁管理的需求，把锁给哪一个客户端又成为一个难点），这个节点又成为单点，岂不是可靠性不行了，失去了分布式多副本的意义，同时性能也很差，另外，还会出现死锁等情况。

所以，说来说去，只有解决分布式条件下的一致性问题，似乎才能解决本质问题。

如上面的例子，Paxos解决这一问题利用的是选举，少数服从多数的思想，只要2N+1个节点中，有N个以上同意了某个决定，则认为系统达到了一致，并且按照Paxos原则，最终理论上也达到了一致，不会再改变。这样的话，客户端不必与所有服务器通信，选择与大部分通信即可；也无需服务器都全部处于工作状态，有一些服务器挂掉，只

Paxos中的Acceptor就相当于上面的队长，Proposer就相当于上面的驴友，epoch编号就相当于例子中申请短信的发送时间。关于Paxos的正式描述已经很多了，这里就不复述了，关于Paxos正确性的证明，因为比较复杂，以后有时间再分析。另外，Paxos最消耗时间的地方就在于需要半数以上同意沟通了才能进入第二步，试想一下，一开始，所有驴友就给队长狂发短信，每个队长收到的最新短信的是不同驴友，这样，就难以达到半数以上都同意与某个驴友沟通的状态，为了减小这个时间，Paxos还有Fast Paxos的改进等等，有空再分析。

倒是有一些问题可以思考一下：在Paxos之前，或者说除了Chubby，ZooKeeper这些系统，其它分布式系统同样面临这样的一致性问题，比如HDFS、分布式数据库、Amazon的Dynamo等等，解决思路又不同，有空再进行对比分析。

分布式理论之一：Paxos算法的通俗理解

0909 发表于1年前

关于Paxos说的一致性，个人理解是指冗余副本（或状态等，但都是因为存在冗余）的一致性。这与关系型数据库中ACID的一致性说的不是一个东西。在关系数据库里，可以连副本都没有，何谈副本的一致性？按照经典定义，ACID中的C指的是在一个事务中，事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。那么，什么又是一致性状态呢，这跟业务约束有关系，比如经典的转账事务，事务处理完毕后，不能出现一个账户钱被扣了，另一个账户的钱没有增加的情况，如果两者加起来的钱还是等于转账前的钱，那么就是一致性状态。

从很多博文来看，对这两种一致性往往混淆起来。另外，CAP原则里面所说的一致性，个人认为是指副本一致性，与Paxos里面的一致性接近。都是处理“因为冗余数据的存在而需要保证多个副本保持一致”的问题，NoSQL放弃的强一致性也是指副本一致性，最终一致性也是指副本达到完全相同存在一定延时。

当然，如果数据库本身是分布式的，且存在冗余副本，则除了解决事务在业务逻辑上的一致性问题外，同时需要解决副本一致性问题，此时可以利用Paxos协议。但解决了副本一致性问题，还不能完全解决业务逻辑一致性；如果是分布式数据库，但并不存在副本的情况，事务的一致性需要根据业务约束进行设计。

另外，谈到Paxos时，还会涉及到拜占庭将军问题，它指的是在存在消息丢失的不可靠信道上试图通过消息传递的方式达到一致性是不可能的。Paxos本身就是利用消息传递方式解决一致性问题的，所以它的假定是信道必须可靠，这里的可靠，主要指消息不会被篡改。消息丢失是允许的。

关于一致性、事务的ACID，CAP，NoSQL等等问题，以后再详细分析。本文的描述也许可能存在一些举例不太恰当的地方，如果错误，欢迎批评指正。

分类：工作日志 字数：4048

标签： Paxos算法

0909

无锡

相关博客

Paxos算法
风林火山
1422 1

分布式–Paxos算法系列文章
小树鹿鸣
18 0

Paxos算法在大型系统中常见的应用场景
wikison
37 0

评论 (0)

Ctrl+Enter 发表评论