Oceanbase列传

分布式与存储技术

[Paxos三部曲之一] 使用Basic-Paxos协议的日志同步与恢复

使用Basic-Paxos协议的日志同步与恢复

在保证数据安全的基础上，保持服务的持续可用，是核心业务对底层数据存储系统的基本要求。业界常见MySQL/Oracle的1主N备的方案面临的问题是"最大可用（Maximum Availability）"和"最大保护（Maximum Protection）"模式间的艰难抉择，其中"最大可用"模式，表示主机尽力将数据同步到备机之后才返回成功，如果备机宕机或网络中断那么主机则单独提供服务，这意味着主备都宕机情况下可能的数据丢失；"最大保护"模式，表示主机一定要将数据同步到备机后才能返回成功，则意味着在任意备机宕机或网络中断情况下主机不得不停服务等待备机或网络恢复。可见传统主备方式下，如果要求数据不丢，那么基本放弃了服务的持续可用能力。

基于Paxos协议的数据同步与传统主备方式最大的区别在与Paxos只需任意超过半数的副本在线且相互通信正常，就可以保证服务的持续可用，且数据不丢失。本文不再分析Paxos协议本身（参考原始论文，以及这篇比较通俗的分析http://mp.weixin.qq.com/s?__biz=MjM5MDg2NjIyMA==&mid=203607654&idx=1&sn=bfe71374fbca7ec5adf31bd3500ab95a&key=8ea74966bf01cfb6684dc066454e04bb5194d780db67f87b55480b52800238c2dfae323218ee8645f0c094e607ea7e6f&ascene=1&uin=MjA1MDk3Njk1&devicetype=webwx&version=70000001&pass_ticket=2ivcW%2FcENyzkz%2FGjIaPDdMzzf%2Bberd36%2FR3FYecikmo%3D ），而是基于Paxos协议，讨论一种在多副本上持久化数据的高可用方案。需要注意的是，本方案不考虑运行性能，只是为了帮助说清协议的工程实现。

我们将数据持久化的需求抽象为：在N个server的机群上，持久化数据库或者文件系统的操作日志，并且为每条日志分配连续递增的logID，我们允许多个客户端并发的向机群内的任意机器发送日志同步请求。对于高可用的需求为：在N个server中只要有超过半数的server（majority）正常服务，并且相互通信正常，那么这个机器就可以持续的提供日志持久化和查询服务。

将每条日志的持久化流程都看作一个"Paxos Instance"，不同的logID代表不同的Paxos Instance形成的"决议（decision）"。即每一个logID标识着一轮完整paxos协议流程的执行，最后形成decision。机群内的每个server同

时作为paxos的acceptor和proposer。

获取LogID

Server收到客户端的持久化日志请求后，先要决定这条日志的logID，为了尽量减少后续Paxos协议流程中处理并发冲突造成的回退，要尽量分配与目前已经持久化和正在持久化中的日志不重复的logID，同步也要容忍少于半数的server宕机与网络故障。因此向所有acceptor查询它们本地目前已写盘的最大logID，而只需收集到majority返回的结果，并选择其中最大的logID+1作为本次待持久化日志的logID。从上面的描述可以看出，这里并不能保证并发提交的两条日志一定被分配到不同的logID，而是依靠后续的paxos协议流程来达到对一个logID形成唯一的decision的目的。

产生ProposalID

获取LogID后，server作为proposer开始针对当前logID，执行Paxos Instance，先产生proposalID，根据paxos协议的要求，proposalID要满足全局唯一和递增序，即对同一个server来说后产生的proposalID一定大于之前产生的，这里我们使用server的timestamp联合ip作为proposalID，其中timestamp在高位，ip在低位，只要时钟的误差范围小于server重启的时间，就可以满足"同一个server后产生的proposalID一定大于之前产生的"。

Prepare阶段

Proposer准备好proposalID后，将proposalID作为 "提案（proposal）"发送给所有的acceptor。根据Paxos协议P1b的约束，这个阶段发送的proposal并不需要携带日志内容，而只需要发送proposalID。Acceptor收到proposal后，根据Paxos协议P1b判断是否要"回应（response）"：只有在这个Paxos Instance内（即针对这个logID）没有response过proposalID大于等于当前proposal的，并且也没有"接受（accept）"过proposalID大于当前proposal的，才可以response，并承诺不再accept那些proposalID小于当前proposal的。

如果已经accept过proposal，那么连同proposalID最大的日志内容一同response。为了遵守P1b的约束，在宕机恢复后也能满足，因此在response前，需要将当前proposalID写到本地磁盘。

上述Prepare阶段的处理流程暗示，对于分配到相同logID的不同日志，由于他们的proposalID不同，acceptor在response一个较小proposalID后，是允许继续response后来的较大的proposalID的。

Accept请求阶段

Proposer收集到majority的response后，来决定后续是否将要发出的"accept请求（accept request）"，判断如果majority的response中的日志内容都为空，那么可以向所有acceptor发出accept request并携带上当前日志

内容；而如果有任意的response中的日志内容有效，那么说明当前logID已经别其他日志占用，且其他日志可能已经在majority上持久化，因此需要回退，回到第一步“获取logID”重新执行。

Accept处理阶段

Acceptor收到proposer的accept request后，根据上文中“Prepare阶段”的承诺，判断当前logID下，曾经response过的最大proposalID，如果小于等于当前proposal的，则可以继续执行后续的accept处理逻辑；而如果大于当前proposal的，则说明有logID切proposalID更大的proposal在并发执行，当前proposal会被覆盖，因此回复proposer要求回退到第一步“获取logID”重新执行。

然后Accept处理逻辑将当前proposal连同proposalID一起写到本地磁盘，给proposer回复成功。Proposer收集到majority的回复成功后，说明本条日志已经在机群上持久化成功，可以保证后续一定不会被覆盖或丢失，可以给客户端返回了。

上述accept处理阶段的流程暗示，可能会存在针对一个logID，日志只在少于半数的acceptor上写到本地磁盘，而acceptor同时response了proposalID更大的proposal，而使得当前logID下没有任何日志在机群上持久化成功。即一个logID可能没有标识任何有效日志，这种情况是可以接**受的。**

日志内容读取

已经在机群上持久化成功的日志，需要能够被读取出来，一般的应用模式是按照logID的顺序依次读取并回放日志。读取的时候针对每一条logID，需要执行一轮完整的paxos协议流程，将accept处理阶段成功的日志内容返回。需要注意的是，在accept请求阶段的处理逻辑变化：Proposer收集到majority的response后，判断如果majority的response中的日志内容都为空，那么向所有acceptor发出日志内容为空的accept request；而如果有任意的response中的日志内容有效，则选择proposalID最大的日志内容放入accept request。后续收到majority的accept回复成功后，才可以返回日志内容作为读取结果。

这里的流程暗示，针对一个logID，如果之前已经有日志内容持久化成功，那么这条日志一定会被选为accept request；而如果之前日志内容仅仅在小于半数的server上写到磁盘，那么最终这条logID的内容有可能是有效日志，也有可能内容为空。

为什么读取也需要执行Paxos流程

这是基于一致性的考虑，即针对一条logID，读取出的内容后续应该永远不变。因此如果一条logID在写入过程中，并未在majority上持久化，那么需要在读取返回结果前，将这个结果在机群上持久化成功。

本条目发布于2015年3月11日 [http://oceanbase.org.cn/archives/90] 。属于paxos、分布式系统分类，被贴了CAP、database、log、paxos、redo 标签。

《[Paxos三部曲之一] 使用Basic-Paxos协议的日志同步与恢复》有25个想法

四不象
2016年2月14日 上午10:34

看完全文，有以下几点不是很明白：
1、『proposalID要满足全局唯一和递增序』。Paxos 的提案编号不应该是全局自增的么？
2、唯一的冲突可能就是logId重复。如果能解决logId按顺序全局自增，似乎不使用Paxos也是可以的。

yubai 文章作者
2016年2月14日 下午9:31

1. Paxos本身并未对proposalID有什么要求，它的协议保证在未形成决议的情况下，一定是proposalID较大的提案形成决议，理论上完全可以随机生成唯一的proposalID，只不过它无法通过P1b约束的概率很高而已。

2. 如果你有办法为每条日志产生全局唯一递增的logid，那确实不需要完整的paxos流程了，每条日志走一遍accept就成了，这个正是multi-paxos的思路，最重要的是你要考虑这个单点的logid生成器怎么实现

四不象
2016年2月15日 上午9:32

明白了，非常感谢

alwens
2016年3月8日 上午2:17

> 赞，目前看过的最接地气的Paxos

frivol
2016年3月13日 下午7:21

日志应该是有顺序的吧？因为日志对应于客户端发来的每一条命令。
那么在“ Accept请求阶段”，当发现 “有任意的response中的日志内容有效”，logID被（假设A）占用，怎么确定当前日志和A表示的日志哪个顺序在前呢？如果当前日志顺序在前，那么重新获取ID，不是大于A了吗？

yubai 文章作者
2016年3月17日 下午10:09

日志间的顺序就是靠这组paxos server来定的，像你举的这个例子，这两条日志肯定是并发过来的，怎么定序是paxos server说了算的；如果真是外部调用有顺序性，那一定是前一条日志都投票成功了，后一条日志过来才能保证后一条的logid更大

zhengran
2016年6月15日 下午3:17

这是基于一致性的考虑，即针对一条logID，读取出的内容后续应该永远不变。因此如果一条logID在写入过程中，并未在majority上持久化，那么需要在读取返回结果前，将这个结果在机群上持久化成功。
>>> 这里没明白, 如果一条logid没有在majority上持久化, 那么有可能读取之后, 这条logid就在majority上持久化成功了, 那这样的话, 用户逻辑不就有问题了吗? 比如用户写一条数据, 给用户返回失败了, 然而过一会再去查询的话, 发现这条数据写入成功了, 这不是矛盾了吗?

yubai 文章作者
2016年6月15日 下午10:31

可能这里没写明白，这里的意思是说，客户端会拿到三种结果，1.Unknown（即不知道这条日志是否提交成功，客户端不能假设成功或失败，只能继续查询）；2.成功；3.失败。对于“一条logid没有在majority上持久化”，这就是

个“中间状态”，只能给客户端返回Unknow。对这个logid，再跑一轮Paxos成功的话，结果要么是把客户端要写的内容同步多数派，要么把一条空日志同步多数派。

zhengran
2016年6月16日 上午8:14

谢谢回复. 能不能再详细解释一下日志读取的过程? 我对paxos的learner过程一直不是很理解, 在没有acceptor通知learner的情况下, learner怎么样才能知道最后被批准的value呢? 除非learner读取到所有acceptor返回的结果, 然后判断哪个value是多数派, 但是如果有些acceptor故障, 无法返回结果呢?

yubai 文章作者
2016年6月16日 下午12:57

读取时，必须保证有多数派的acceptor能返回结果才行哦

zhengran
2016年6月16日 下午1:26

即使有多数派返回了结果, 也只能断定多数派里包含了被批准的value, 但是仍然无法确定哪个是被批准的value吧? 怎么样才能确定被批准的value呢?

yubai 文章作者
2016年6月20日 下午2:53

可以的，请仔细琢磨下P2a中，对proposal内容的选择规则

zhengran
2016年6月21日 上午8:43

假设有a, b, c 3个节点, 刚开始的时候, 3个节点的值是一致的, 都是v1. 这时候一个proposer发起一轮paxos请求, 希望将值设置成v2. 那么可能出现两种情况, 一种是paxos成功了, a, b, c的多数派中value设置成了v2, 也就是说v2被批准了(假设a, b, c的值分别是v1, v2, v2). 另外一种是失败了, a, b, c的多数派中value仍然是v1, 也就是说v2没有被批准, 但是a, b, c中可能存在少数派节点的值是v2(假设a, b, c的值分别是v1, v1, v2). 按照p2a的流程, 选取proposal的原则是选取多数派中, proposal号最大的作为期望被批准的value. 在上述两种情况下, 当reader读取数据时, 按照p2a的逻辑, 既可能读到v1, 也可能读到v2, 那这样的话, 如果有一个acceptor故障了, 其实reader就无法准确知道之前被批准的value了吧? 我看paxos made simple的论文中也提到, 如果有一个acceptor故障了, 也不可能找出被批准的value了, 唯一的办法就是issue a proposal, 然后运行一轮新的paxos算法了. 这是不是说, 不管reader读到的是v1还是v2, 仍然发起一轮paxos协议, 保证一致性就ok了, 而不需要考虑v2是否被批准了?

yubai 文章作者
2016年6月27日 下午1:02

非常正确！读取出V1，就是通过issue a proposal来做到的，这样v1这个值的proposal id就会大于v2的，后续如果再执行paxos，根据P2a，将一直得到v1

林泽伟
2016年8月15日 下午8:19

用Paxos协议的日志同步系统同一时间只能有一条日志在写？这样的并发量较低

yubai 文章作者
2016年8月19日 下午2:10

group commit把多条日志放在一次paxos提交中；并且相对于raft，multi-paxos允许多个日志一起提交，不需要等上一个成功

刘超
2016年8月26日 下午5:10

这里的流程暗示，针对一个logID，如果之前已经有日志内容持久化成功，那么这条日志一定会被选为accept request；而如果之前日志内容仅仅在小于半数的server上写到磁盘，那么最终这条logID的内容有可能是有效日

志，也有可能内容为空。

请问对于小于半数写入磁盘的日志，并没有处理成功，后续还会重新选择logID进行重试，那读出时会不会出现重复?

yubai 文章作者
2016年9月12日 下午10:49

其实并不会，因为“小半数”成功的情况下，对于上层应用来说得到的并非“写失败”而是unknow，对于数据库来讲这种情况下，它没有别的选择，只有退出，等到新leader对未决日志重新投票

sainth
2016年8月26日 下午7:22

我看了三天各种博客，
本文是唯一一个讲明了如何使用Paxos算法的（即什么是一轮Paxos，什么是提议），并且例子基本是复核paxos made simple 论文的。
唯一的问题是：按照这种例子，当某个提案者发现accepter返回说允许，但是已经接收过某个提案，其值是A时，原Paxos算法认为：该提案者继续进入下一阶段，但是将值换为A。
本例子认为：提案者可以停了，结束概论Paxos，去寻找新的LogID。

以上我认为是有道理的，但是和Paxos算法的描述有点出入。

yubai 文章作者
2016年9月12日 下午10:52

谢谢你的关注!

是的，paxos原始协议并没有说该如何处理redolog连续写的问题，这里的方式与megastore论文中暗示的方法时比较相似的，相当于扩展了paxos协议，对原始算法没有侵入

luhongpeng
2016年8月30日 上午11:35

日志内容可不可以是某个文件（小文件）？就是说将某个文件的内容当做paxos的value来propose，一旦commit成功后，就可以在认为它在集群里的大多数节点上的值是一致的了？

yubai 文章作者
2016年9月12日 下午10:53

当然可以，你只需要用一个唯一的key来标记这个paxos instance就行

mifan
2016年10月24日 下午9:45

上述accept处理阶段的流程暗示，可能会存在针对一个logID，日志只在少于半数的acceptor上写到本地磁盘，而acceptor同时response了proposalID更大的proposal，而使得当前logID下没有任何日志在机群上持久化成功。即一个logID可能没有标识任何有效日志，这种情况是可以接受的。

请教一下，如果有更大的proposalId存在，那么这个proposalId在这个logId对应的肯定有超过半数成功，怎么会有没有任何有效日志?

yubai 文章作者
2016年11月12日 下午7:21

这里没有写清楚，是在一个log id上，有两个proprosal在竞争，但是谁也没有得到多数派，这种情况下，他们充实时可能会用上更大的log id了

小酒馆老板
2018年7月31日 上午12:09

> ## Accept
> 而如果有任意的response中的日志内容有效，那么说明当前logID已经别其他日志占用，且其他日志可能已经在majority上持久化，因此需要回退，回到第一步“获取logID”重新执行。

在这里，我看到的是，如果我提交的prepare提案被拒绝了，我依旧重试我的logid，并且自增.

但是，我在![paxos wiki](https://en.wikipedia.org/wiki/Paxos_(computer_science))上看到的是说，因为你提交的propseid大于它所记录的id，他会把这条ID记录，并且将他之前记录的最大的提案号的值返回。因为他将你的最新的这条提案的id记录了，会导致，下次本来可以提交accpet的propose提交的数据出现错误，按照咱们写的 他不又再次回到 获取 logid 阶段了吗？ 这岂不是说？ 只要使用 basic-paxos，就一定会死锁？？

另外咱们写到 prepare阶段，只需要发送proposeid，不需要协议内容。那么，我作为第一个提交的proposeid，accpter只记录了我的id啊？如果遇到了第二个提案，他不是该告诉第二个提案，我的内容吗？但是我没带啊？