Oceanbase列传

分布式与存储技术

[Paxos三部曲之二] 使用Multi-Paxos协议的日志同步与恢复

使用Multi-Paxos协议的日志同步与恢复

本文是Paxos三部曲中的第二篇。在前一篇文章《使用Basic-Paxos协议的日志同步与恢复》（http://oceanbase.org.cn/?p=90）中，我们讨论了基于Basic-Paxos协议的日志同步方案，在这个方案中，所有成员的身份都是平等的，任何成员都可以提出日志持久化的提案，并且尝试在成员组中进行持久化。而在实际的工程应用中，往往需要一个成员在一段时间内保持唯一leader的身份，来服务对数据的增删改操作，产生redolog，并尝试在成员组中进行持久化。本文讨论如何利用Paxos协议选举唯一的leader，以及使用leader将redolog在成员组中进行持久化和恢复的方法。

- **Basic-Paxos协议回顾**

让我们先来回顾下Basic-Paxos协议的执行流程：为了简化描述，我们假设一个Paxos集群，每个server同时担任proposer和acceptor，任何server都可以发起持久化redolog的请求，首先要向所有的server查询当前最大logID，从多数派的应答结果中选择最大的logID，加1后作为执行本次Paxos Instance的唯一标识；然后进入Paxos的prepare阶段，产生proposalID，并决定出要投票的redolog（即议案）；在accept阶段对prepare阶段产生的议案进行投票，得到多数派确认后返回成功。由此我们可以看出Basic-Paxos协议的执行流程针对每条redolog都至少存在三次网络交互的延迟（1. 产生logID；2. prepare阶段；3. accept阶段）。下面我们逐个分析每个阶段的必要性：

1. 产生logID，由于Basic-Paxos并不假设一段时间内只有唯一的proposer，因此可能由集群内的任意server发起redolog同步，因此不能由单一server维护logID，而是需要进行分布式协商，为不同的redolog分配全局唯一且有序的logID。
2. prepare阶段，上述一阶段的分布式协商logID并不能保证并发多个server分配得到得logID是唯一的，即会出现若干条不同的redolog在同一个Paxos Instance中投票的情况，而这正是Basic-Paxos协议的基本假设，因此需要执行prepare，以决定出这个Paxos Instance内的要进行投票的redolog（即议案）。如果执行prepare决定出的议案与server自己要投票的redolog内容不同，则需要重新产生logID。
3. accept阶段，对prepare阶段决定出的议案进行投票，得到多数派确认后表示redolog同步成功，否则需要重新产生logID。

在这三个阶段中，根据Paxos协议的约束，server应答prepare消息和accept消息前都要持久化本地redolog，以避免重启后的行为与重启前自相矛盾。因此最终可以得到使用Basic-Paxos进行redolog同步的延迟包括了3次网络交互加2次写本地磁盘。并且在高并发的情况下，不同redolog可能被分配到相同的logID，最差可能会在accept阶段才会失败重试。

- **Multi-Paxos协议概述**

在Paxos集群中利用Paxos协议选举唯一的leader，在leader有效期内所有的议案都只能由leader发起，这里强化了协议的假设：即leader有效期内不会有其他server提出的议案。因此对于redolog的同步过程，我们可以简化掉产生logID阶段和prepare阶段，而是由唯一的leader产生logID，然后直接执行accept，得到多数派确认即表示redolog同步成功。

- **leader的产生**

首先，需要明确的是Multi-Paxos协议并不假设全局必须只能有唯一的leader来生成日志，它允许有多个“自认为是leader的server”来并发生成日志，这样的场景即退化为Basic-Paxos。

Multi-Paxos可以简单的理解为，经过一轮的Basic-Paxos，成功得到多数派accept的proposer即成为leader（这个过程称为leader Elect），之后可以通过lease机制，保持这个leader的身份，使得其他proposer不再发起提案，这样就进入了一个leader任期。在leader任期中，由于没有了并发冲突，这个leader在对后续的日志进行投票时，不必每次都向多数派询问logID，也不必执行prepare阶段，直接执行accept阶段即可。

因此在Multi-Paxos中，我们将leader Elect过程中的prepare操作，视为对leader任期内将要写的所有日志的一次性prepare操作，在leader任期内投票的所有日志将携带有相同的proposalID。需要强调的是，为了遵守Basic-Paxos协议约束，在leader Elect的prepare阶段，acceptor应答prepare成功的消息之前要先将这次prepare请求所携带的proposalID持久化到本地。

对于leader Elect过程，我们并不关心leader Elect提案和决议的具体内容，因为无论执行多少次leader Elect，从Basic-Paxos的角度来看，都是同一个Paxos Instance在对已经形成的决议反复进行投票而已。而执行leader Elect这个过程，我们最关注的是要得到最近一次形成决议的proposer是谁，以及它的proposalID。在leader Elect过程中，得到多数派accept的proposer将成为leader，而它本次所用的proposalID即成为它任期内对所有日志（包括新增日志和后文将提到的重确认日志）进行投票时将要使用的proposalID（称为leader ProposalID）。

这里还需要考虑的一个问题是，由于多个server并发执行leader Elect，可能出现两个server在相近的时间内，先后执行leader Elect都成功，都认为自己是leader的情况。因此，当选leader在开始以leader身份提供服务之前，要使用leader ProposalID写一条日志（称为StartWorking日志），得到多数派确认后，再开始提供服务。这是因为根据Basic-Paxos的约束，可以推断出：先执行leader Elect成功的leader（称为L1），它的proposalID（称为P1）一定会小于后执行leader Elect成功的leader（称为L2）的proposalID（称为P2），而经过了两轮leader Elect，机群内多数派持久化的proposalID一定是P2，而此时L1使用P1执行accept时，由于P1<P2，它将无法得到机群内多数派的accept。

- **Confirm日志的优化**

在Paxos协议中，对于决议的读取也是需要执行一轮Paxos过程的，在实际工程中做数据恢复时，对每条日志都执行一轮Paxos的代价过大，因此引入需要引入一种被成为confirm的机制，即leader持久化一条日志，得到多数派的accept后，就再写一条针对这条日志的confirm日志，表示这条日志已经确认形成了多数派备份，在回放日志时，判断如果一条日志有对应的confirm日志，则可以直接读取本地内容，而不需要再执行一轮Paxos。confirm日志只要写本地即可，不需要同步到备机，但是出于提示备机及时回放收到日志的考虑（备机收到一条日志后并不能立即回放，需要确认这条日志已经形成多数派备份才能回放），leader也会批量的给备机同步confirm日志。出于性能的考虑，confirm日志往往是延迟的成批写出去，因此仍然会出现部分日志已经形成多数派备份，但是没有对应的confirm日志的情况，对于这些日志，需要在恢复过程中进行重确认。

在实际的工程实践中，可以使用基于logID的滑动窗口机制来限制confirm日志与对应的原始日志的距离，以简化日志回放与查询逻辑。

- **新任leader对日志的重确认**

如上一节所述，在恢复过程中，拥有对应confirm日志的原始日志，可以被直接回放。而没有对应confirm日志的原始日志，则需要执行一轮Paxos，这个过程被成为重确认。

此外日志中的“空洞”，也需要进行重确认，因为当前leader再上一任leader的任期内可能错过了一些日志的同步，而这些日志在其他机器上形成多了多数派。由于logID连续递增，被错过的日志就成了连续logID连续递增序列中的“空洞”，需要通过重确认来补全这些“空洞”位置的日志。

新任leader在开始执行重确认前，需要先知道重确认的结束位置，因为leader本地相对于集群内多数派可能已经落后很多日志，所以需要想集群内其他server发送请求，查询每个server本地的最大logID，并从多数派的应答中选择最大的logID作为重确认的结束位置。也即开始提供服务后写日志的起始logID。

对于每条日志的重确认，需要执行一轮完整的Paxos过程，可能有些日志在恢复前确实未形成多数派备份，需要通过重新执行Paxos来把这些日志重新持久化才能回放。这种不管日志是否曾经形成多数派备份，都重新尝试持久化的原则，我们称之为“最大commit原则”。之所以要遵守“最大commit原则”，是因为我们无法区分出来未形成多数派备份的日志，而这些日志在上一任leader任期内，也必然是“未决”状态，尚未应答客户端，所以无论如何都重新持久化都是安全的。比如A/B/C三个server，一条日志在A/B上持久化成功，已经形成多数派，然后B宕机；另一种情况，A/B/C三个server，一条日志只在A上持久化成功，超时未形成多数派，然后B宕机。上述两种情况，最终的状态都是A上有一条日志，C上没有，在恢复时无法区分这条日志是否曾经形成过多数派，因此干脆按照“最大commit原则”将这条日志尝试重新在A/C上持久化后再回放。

需要注意的是，重确认日志时，要使用当前的leader ProposalID作为Paxos协议中的proposalID来对日志执行Paxos过程。因此在回放日志时，对于logID相同的多条日志，要以proposalID最大的为准。

- **“幽灵复现”日志的处理**

使用Paxos协议处理日志的备份与恢复，可以保证确认形成多数派的日志不丢失，但是无法避免一种被称为“幽灵复现”的现象，如下图所示：

	Leader	A	B	C
第一轮	A	1-10	1-5	1-5
第二轮	B	宕机	1-6,20	1-6,20
第三轮	A	1-20	1-20	1-20

1. 第一轮中A被选为Leader，写下了1-10号日志，其中1-5号日志形成了多数派，并且已给客户端应答，而对于6-10号日志，客户端超时未能得到应答。
2. 第二轮，A宕机，B被选为Leader，由于B和C的最大的logID都是5，因此B不会去重确认6-10号日志，而是从6开始写新的日志，此时如果客户端来查询的话，是查询不到6-10号日志内容的，此后第二轮又写入了6-20号日志，但是只有6号和20号日志在多数派上持久化成功。
3. 第三轮，A又被选为Leader，从多数派中可以得到最大logID为20，因此要将7-20号日志执行重确认，其中就包括了A上的7-10号日志，之后客户端再来查询的话，会发现上次查询不到的7-10号日志又像幽灵一样重新出现了。

对于将Paxos协议应用在数据库日志同步场景的情况，**“幽灵复现”**问题是不可接受，一个简单的例子就是转账场景，用户转账时如果返回结果超时，那么往往会查询一下转账是否成功，来决定是否重试一下。如果第一次查询转账结果时，发现未生效而重试，而转账事务日志作为幽灵复现日志重新出现的话，就造成了用户重复转账。

为了处理**“幽灵复现”问题，我们在每条日志的内容中保存一个generateID，leader在生成这条日志时以当前的leader ProposalID作为generateID。按logID顺序回放日志时，因为leader在开始服务之前一定会写一条StartWorking日志，所以如果出现generateID相对前一条日志变小的情况，说明这是一条“幽灵复现”日志（它的generateID会小于StartWorking日志），要忽略掉这条日志。**

- **总结**

本文介绍了在Basic-Paxos协议基础之上构建Multi-Paxos协议的几个要点：通过使用Paxos选举leader来避免对每条日志都执行Paxos的三阶段交互，而是将绝大多数场景简化为一阶段交互，并且讨论了基于Paxos协议的“最大commit原则”；通过引入confirm日志来简化回放处理；通过引入Start Working日志和generateID来处理“幽灵复现”问题。

本条目发布于2015年12月14日 [http://oceanbase.org.cn/archives/111] 。属于paxos、分布式系统分类，被贴了paxos 标签。

《[Paxos三部曲之二] 使用Multi-Paxos协议的日志同步与恢复》有11个想法

包增辉
2016年3月4日 下午7:41

请教一下：我觉得不会出现"幽灵日志"的情况，因为 一般leader写日志的请求处理流程是：leader发命令到slave机器，并等待slave返回 –> slave 持久化数据，并返回ack. –> leader 收集到 过半数的ack后，再写ComfirmID、返回ack给用户、持久化到磁盘。
所以不会出现 第一轮 "leader的日志比 多数slave 落后" 的那种场景啊。

yubai 文章作者
2016年3月5日 上午2:10

leader自己写，发给slave都是异步的，谁先谁后是不确定的

alwensohng
2016年3月9日 下午4:55

请问幽灵现象是否只发生在，只在本机持久化成功，然后本机宕机一会突然又恢复的情况?

假设有五台机器 ABCDE， A为leader, 某个logid 在 AB上持久化成功，然后A宕机一会恢复，虽然这种情况也是未得到majority的持久化，但感觉这种情况是不会幽灵现象的，因为最大提交原则的存在，会使B上的那份数据提交到majority中。不知道理解得对不。

yubai 文章作者
2016年3月9日 下午8:47

你这个case确实不会出现幽灵复现的问题，幽灵复现问题的根本原因是leader的乱序提交log

frivol
2016年3月20日 下午10:15

提两个问题：
1. 对于multi-paxos,客户端是否只知道机群中每个成员的地址列表？那是怎么把命令交给leader的呢？
2. 设想有这样一种方式，跟paxos完全没有关系，你看能否构成一种实现主备机一致的方案呢？如果不行，漏洞在哪呢？
0）多播可以组成一个机群的组。
1）如果机群中每个机器都维护一个所有成员的有序列表，大家都认为列表排在第一的成员为leader不就不用选举了吗？
2） leader只需向多数派的机器同步完日志，就可以应答客户端了，对于其他的备机，可以应答客户端之后向其异步发送日志即可。
3） 失败后醒来重新加入机群的机器向leader要它缺失的日志，不就可以完成日志同步了吗？

谢谢。

yubai 文章作者
2016年3月23日 上午10:12

1. 客户端持有的成员组信息是不保证最新的，一般会拿着所有成员的地址，把请求发给一个非leader的成员时，这个成员会把它认为的当前leader地址返回给客户端
2. 你的这个方案如果leader跟少数派成员网络分区了，新的leader怎么选？

frivol
2016年3月23日 下午8:56

1. 正常的paxos，ABCDE，A是leader，如果AB与CDE分区了，客户端恰好发给A或B，这时整个系统是怎样实现重新走上正轨完成工作的呢？

2. 这个方案中，假设也是ABCDE，如果发生分区，约定每个机器发现当前集群总数小于多数派的数量，就回答客户端，机器数量过少，无法工作。我想这时客户端应该尝试向列表中其他机器发送请求，看能否得到回应。而多数派CDE按照集群中机器的顺序，AB都离开了，当然都认为C是leader了，它们还可以达到多数派，继续提供服务。

yubai 文章作者
2016年3月24日 上午11:19

1. 客户端发给A/B，会得到not leader的回复，这样客户端就要尝试去连接CDE
2. “约定每个机器发现当前集群总数小于多数派的数量”，这个是做不到的，paxos的根本在于一个成员不能做出决策，一定是多数派才能做决策，所以你这个回复客户端“机器数量过少，无法工作”这个决策是不能做出来的。
你的意思就是：比如作为D，如果AB连接不上，我就认为C是leader对吧；那么作为C如果AB连接不上，我认为自己是leader，这样明显不对嘛

frivol
2016年3月27日 下午12:06

当集群成员发生变化时（加入，离开），每个成员会得到通知，所以可以知道当前成员数量的。
假如AB单独分区了，那么在这个子分区内，A是leader，这个分区成员数量小于多数派，已经不能工作了，但另一个子分区CDE中，C是leader，这没错啊，CDE每个人都认为C是leader，你说”这样明显不对嘛“ 是什么意思呢?

Jesse
2016年10月18日 上午11:24

你说的这个类似 Zookeeper Atomic Broadcast，选最大ID的为主。
但是 ZAB主崩溃后，进入 discovery 阶段， followers 会给剩下机器中ID最大的（ prospective leader ）发主挂消息，当 prospective leader 收到 大多数 followers 的主挂消息后，他就认为自己应该为主了，并给 followers 发送选自己为主的消息，当且仅当 prospective leader 得到大多数 followers 的 ack 后才确认自己 leader 地位。进入 sync 阶段。

细节可以参考 https://blog.acolyer.org/2015/03/09/zab-high-performance-broadcast-for-primary-backup-systems/

frivol
2016年3月21日 下午9:01

最后加入的机器被加入列表的尾部。所以，leader一定是当前所有存活的机器中存活得最久的一个。