Oceanbase列传

分布式与存储技术

[Paxos三部曲之三] Paxos成员组变更

Paxos成员组变更

本文是Paxos三部曲的第三篇，在前一篇文章《使用Multi-Paxos协议的日志同步与恢复》（http://oceanbase.org.cn/?p=111）中，我们讨论了基于Multi-Paxos协议的日志同步方案，在这个方案中，我们有一个隐含的前提，就是Paxos成员组是确定的，并且所有成员启动后都能加载一致的成员组信息。而在实际的工程应用中，往往需要在不停服务的情况下修改成员组，最典型的比如类似spanner的系统，对子表的迁移操作过程，就包含了对其Paxos成员组的变更操作。本文将基于Raft论文，讨论通用的成员组变更方法，和简化的一阶段成员组变更方法，以及成员组变更与日志同步操作的关系。

请注意，本文假设读者已了解Basic-Paxos和Multi-Paxos协议，并且本文假设集群工作在上一篇文章所述的Multi-Paxos协议之下。

- ***在线成员组变更的难点***

Paxos执行两阶段投票协议的前提是有一个明确的Paxos成员组，而对于完全无中心化的Paoxs协议来说，成员组的内容本身又需要通过Paxos协议来维护一致性。对于变更后的新成员组从什么时机开始生效，存在“先有鸡还是先有蛋”的问题，如果还像同步普通日志一样来同步新成员组，那么在新旧成员组交接的过程中宕机，则可能出现选票分裂的情况，比如由成员组ABC变更为ABCDE过程中宕机，AB未持久化新成员组，CED已持久化新成员组，那么在宕机重启后，会出现AB形成了旧成员组的多数派，而CDE形成了新成员组的多数派，会出现两个leader的情况。

因此我们可以总结对在线成员组变更方案的几个基本要求：

P1. 成员组正常Paxos日志同步服务不中断

P2. 任何情况下宕机都能够保证存活的多数派成员间能够选举leader

P3. 不会出现1个以上的多数派选出大于1个leader的情况

- ***成员组变更的基本思路***

成员组变更代表了“旧朝代”的结束和“新朝代”的开启，可以理解为依次执行如下两个投票操作：

Pa. “旧朝代”的多数派成员对“旧朝代结束”这件事达成一致，达成一致后旧成员组不再投票

Pb. “新朝代”的多数派成员对“新朝代开启”这件事达成一致，达成一致后新成员组开始投票

但是简单的按照这种两阶段的操作进行成员变更，虽然能够保证上述P3的约束，但是无法满足P1和P2，比如Pa执行成功后，在Pb执行成功之前：没有成员组可以投票，服务会中断；如果集群宕机重启，新的成员组的各个成员由于还未对新成员组达成一致，而无法选出leader。

为了保证P1和P2的约束，我们在上述基本成员变更的基础上，将Pa和Pb合并为一步操作，即新旧成员组一起对“旧朝代结束+新朝代开启”这件事达成一致后，才表示成员组变更成功。在开始成员变更的投票后，集群就进入了一个“中间状态”，在这个过程中宕机恢复后可能退回“旧朝代”也可能进入“新朝代”，因此在这个中间状态过程中投票的日志，要求在新旧成员组中都达成一致。

在这个基本思路的指导下，可以抽象出一个通用的成员变更方法：Jonit-Consensus。

- ***通用成员组变更方法–Joint-Consensus***

Joint-Consensus是Raft论文中提到的两阶段成员变更方案，这个方案比较通用，甚至可以做到完整的成员组替换，但是两阶段方案的工程实现都比较复杂，而通用的场景需求又不多，因此在他博士论文最终版的成员变更一章中，更多篇幅分析了简化的一阶段方案（下一节讨论），而把Joint-Consensus的篇幅省略了很多。但是作为成员变更方案的基础，我这里还是希望能够从Joint-Consensus开始，分析它的正确性，并且尝试推导出一阶段的成员变更方法。

Joint-Consensus的方案如下，设成员变更前的成员组为C(old)，变更后的成员组为C(new)，成员组内容中包含单调增长的Version。

- **变更操作**

1. 成员变更操作前，C(old)的多数派中持久化的成员组为[[C(old)]]
2. 成员变更操作由leader执行，leader收到命令后，将成员组[[C(old),C(new)]]发送给C(old)∪C(new)的所有成员，在此之后新的日志同步需要保证得到C(old)和C(new)两个多数派的确认
3. leader收到C(old)和C(new)两个多数派确认后，将成员组[[C(new)]]发送给C(new)的所有成员，收到C(new)多数派确认后，表示成员变更成功，后续的日志只要得到C(new)多数派确认即可

- **协议约束**

1. Version投票约束：持有Version较大的成员，不能给持有Version较小的候选人投票
2. 最大commit原则：

- 持有[[C(old),C(new)]]的成员当选leader后，要重新对[[C(old),C(new)]]分别在C(old)和C(new)内投票达成多数派，然后继续成员变更流程，对[[C(new)]]在C(new)内投票达成多数派。然后才能开始leader恢复流程和leader服务
- 持有[[C(old)]]的成员当选leader后，要重新对[[C(old)]]在C(old)内投票达成多数派，然后才能开始leader恢复流程和leader服务
- 持有[[C(new)]]的成员当选leader后，要重新对[[C(new)]]在C(new)内投票达成多数派，然后才能开始leader恢复流程和leader服务

- **选主投票原则**

1. 持有[[C(old),C(new)]]的候选人要得到C(old)和C(new)两个多数派都确认，才能当选leader
2. 持有[[C(old)]]的候选人要得到C(old)多数派确认，才能当选leader
3. 持有[[C(new)]]的候选人要得到C(new)多数派确认，才能当选leader

- **Joint-Consensus的协议分析**

1. 成员变更过程中，对[[C(old),C(new)]]的投票要求在C(old)和C(new)中都得到多数派的确认，是为了保证在C(old)投票“旧朝代结束”成功的同时，“新朝代开启”能够在C(new)生效，不会出现服务中断或者宕机重启后无法选出leader的情况。
2. 对于成员变更的第二步，在[[C(old),C(new)]]形成两个多数派确认后，还要对[[C(new)]]在C(new)中进行投票，是为了结束需要向C(old)和C(new)都同步数据的“中间状态”。[[C(new)]]得到C(new)的多数派确认后，由于后面将要提到的“Version投票约束”原则的保证，可以确保后续宕机重启只有C(new)中的成员能够当选leader，因此无需再向C(old)同步数据。
3. Version投票约束，实际上是Paxos协议Prepare阶段对ProposalID的约束，如本系列的前一篇Multi-Paxos一文所述，选主过程本质上是Paoxs的Prepare过程，我们将成员组内容视为Paxos提案，那么Version就是ProposalID，Paxos不允许Prepare阶段应答ProposalID更低的提案，所以我们要求持有较大Version的成员不能给持有较小Version的候选人投票。从直观上来分析，Version投票约束可以保证，在[[C(new)]]形成多数派确认后，C(old)中那些错过了成员变更日志的成员，不可能再得到C(old)多数派的选票。
4. 最大commit原则，是Paxos最重要的隐含规则之一，在成员变更过程中的宕机重启，持有[[C(old),C(new)]]的成员可能当选leader，但是[[C(old),C(new)]]可能并未形成多数派，根据成员变更协议，成员变更过程要在[[C(old),C(new)]]形成两个多数派确认后，才能对[[C(new)]]进行投票。否则如果立即对[[C(new)]]进行投票，宕机重启后，可能出现C(old)和C(new)两个投票组各自选出一个leader。因此，持有[[C(old),C(new)]]的成员当选leader后，无论[[C(old),C(new)]]是否已经形成两个成员组的多数派确认，我们都按照最大commit原则对它重新投票确认形成多数派后，才能继续leader后续的上任处理。
5. 选主投票原则，持有[[C(old),C(new)]]的成员当选leader，需要得到C(old)和C(new)两个多数派都确认，是为了避免C(old)与C(new)各自形成多数派选出两个leader的情况。在成员变更过程中，可以归结为如下两种情况：

- 对[[C(old),C(new)]]的投票已开始，但未形成两个多数派确认，集群宕机。那么重启选主时，要么持有[[C(old)]]的成员当选leader，要么持有[[C(old),C(new)]]的成员当选leader。
- 对[[C(new)]]的投票已开始，但未形成多数派确认，集群宕机。那么重启选主时，要么持有[[C(new)]]的成员当选leader，要么持有[[C(old),C(new)]]的成员当选leader。

如上文所述，持有[[C(old),C(new)]]的leader要先完成成员变更流程。之后再执行Multi-Paxox中的日志“重确认”，因此日志“重确认”过程不会进入“要得到两个成员组确认”的情况。

Joint-Consensus允许C(old)与C(new)交集为空，在这种情况下成员变更后，旧leader要卸任，并且将leader权限转让给确认[[C(new)]]的一个多数派成员。

Joint-Consensus方案比较通用且容易理解，但是实现比较复杂，同时两阶段的变更协议也会在一定程度上影响变更过程中的服务可用性，因此我们期望增强成员变更的限制，以简化操作流程，考虑Joint-Consensus成员变更，之所以分为两个阶段，是因为对C(old)与C(new)的关系没有做任何假设，为了避免C(old)和C(new)各自形成多数派选出两个leader，才引入了两阶段方案。因此如果增强成员组变更的限制，假设C(old)与C(new)任意的多数派交集不为空，这两个成员组就无法各自形成多数派，那么成员变更方案就可能简化为一阶段。

- ***一阶段成员变更方法***

Raft作者在他博士论文最终版的成员变更一章中，简化了Joint-Consensus的篇幅，而着重介绍了一阶段的成员变更方法，在工程上一阶段的成员变更方法确实更简单实用，下面是我对一阶段成员变更方案的一些分析。

- **每次只变更一个成员**

如上一节所述，如果做到C(old)与C(new)任意的多数派交集都不为空，那么即可保证C(old)与C(new)无法各自形成多数派投票。方法就是每次成员变更只允许增加或删除一个成员。假设C(old)的成员数为N，分析如下：

- C(new)成员数为N+1
    1. 假设选出的leader持有C(new)，那么一定是C(new)中有多数派，即(N+1)/2+1的成员给leader投票，那么持有C(old)且未给leader投票的成员最多为(N+1)-((N+1)/2+1)=(N-1)/2，这个值小于C(old)的多数派值N/2+1，无法选出leader
    2. 假设选出的leader持有C(old)，那么一定是C(old)中有多数派，即N/2+1的成员给leader投票，那么持有C(new)且未给leader投票的成员最多为(N+1)-(N/2+1)=N/2，这个值小于C(new)的多数派值(N+1)/2+1，无法选出leader
- C(new)成员数为N-1
    1. 假设选出的leader持有C(new)，那么一定是C(new)中有多数派，即(N-1)/2+1的成员给leader投票，那么持有C(old)且未给leader投票的成员最多为N-((N-1)/2+1)=(N-1)/2，这个值小于C(old)的多数派值N/2+1，无法选出leader
    2. 假设选出的leader持有C(old)，那么一定是C(old)中有多数派，即N/2+1的成员给leader投票，那么持有C(new)且未给leader投票的成员最多为N-(N/2+1)=(N-2)/2，这个值小于C(new)的多数派值(N-1)/2+1，无法选出leader

- **启用新成员组的时机**

启用新成员组的时机是指从何时开始，对日志的投票开始使用C(new)进行，这里需要考虑的问题是成员变更过程中宕机，重启选主后，持有[[C(old)]]的成员被选为leader，在宕机前使用C(new)同步的日志是否可能丢失。分析如下几种情况：

1. 下线成员，C(new)与C(old)多数派成员数相同，比如ABCDE变更为ABCD，C(new)的任意多数派集合一定是C(old)的某个多数派，变更过程中使用C(new)同步的日志，在C(old)中依然能够保持多数派。
2. 下线成员，C(new)的多数派成员数小于C(old)，比如ABCD变更为ABC，这个情况比较特殊，我们来仔细分析，这种情况下在C(new)中形成的多数派成员只能达到C(old)成员数的一半，从严格的Basic-Paxos协议来分析，只做到N/2的成员确认，是不能保证决议持久化的。但是我们放在Multi-Paxos的环境中，使用lease机制保证leader***有效***（leader"有效"的意思是：StartWorking日志已形成多数派，且完成日志"重确认"，参考上一篇《使用Multi-Paxos协议的日志同步与恢复》）的前提下，因为不会有1个以上的成员并发提出议案，同时又因为在N为偶数时，N/2的成员集合与N/2+1的成员集合的交集一定不为空，可以分析出：在leader ***有效*** 的前提下，只要N/2（N为偶数）的成员确认，即可保证数据持久化。因此，在这种情况下，在C(new)形成多数派的日志，宕机重启后，在C(old)中可以被多数派"重确认"，不会丢失。
3. 上线成员，C(new)的多数派成员数大于C(old)，比如ABC变更为ABCD，C(new)的任意多数派集合一定包含了C(old)的某个多数派，变更过程中使用C(new)同步的日志，在C(old)中依然能够保持多数派。
4. 上线成员，C(new)与C(old)多数派成员数相同，比如ABCD变更为ABCDE，某些情况下可能产生C(new)的多数派（如ABE）与C(old)的多数派（如AB）交集只达到C(old)的一半，情况与第2点相同。

- **最大commit原则**

这里的最大commit原则体现在，同步[[C(new)]]的过程中集群宕机，持有[[C(new)]]的成员当选leader，重启后无法确认当前多数派持有的成员组是[[C(new)]]还是[[C(old)]]，需要leader将当前持有的成员组重新投票形成多数派确认后，才能开始leader后续的上任处理。否则可能出现连续变更情况下，成员组分裂选出2个leader的情况，如Raft报出的这个bug，https://groups.google.com/forum/#!topic/raft-dev/t4xj6dJTP6E，修正方法也很简单就是实用最大commit原则，对成员组重新投票得到多数派确认。

- **阶段成员变更方案总结**

1. 成员变更限制每次只能增加或删除一个成员
2. 成员变更由有效的leader发起，确认新的成员组得到多数派确认后，返回成员变更成功
3. 一次成员变更成功前不允许开始下一次成员变更,因此新任leader在开始提供服务前要将自己本地保存的最新成员组重新投票形成多数派确认
4. leader只要开始同步新成员组后，即可开始使用新的成员组进行日志同步
5. 成员组实用Version标记，持有更大Version的成员不能给持有较小Version的成员投票

成员组变更与日志同步

- **Log Barrier**

对于下线成员的场景，我们需要保证所有日志在剩余在线的机器上能够形成多数派备份，否则可能丢失日志。比如下面的场景，logID为2的日志，在连续成员变更后，仅A上有，无法在A/B/C上形成多数派:

Leader	A	B	C	D	E
A	1,2	1	1	1,2	1,2
A	1,2,Cnew1	1,Cnew1	1,Cnew1	1,2	下线
A	1,2,Cnew1,Cnew2	1,Cnew1,Cnew2	1,Cnew1,Cnew2	下线	下线
	Cnew1=[A,B,C,D] Cnew2 = [A,B,C]				

因此我们要求leader在持久化新的成员组时，要像普通日志一样为它分配logID（称为成员变更日志），它是一个“单向barrier”，即要求所有成员保证logID小于它的日志都持久化本地后，才能持久化成员变更日志，而logID大于它的日志则不受此约束。在上面的例子中,要求B/C保证在持久化 Cnew1之前,一定先保证2号日志持久化。

本条目发布于2016年4月12日 [http://oceanbase.org.cn/archives/160] 。属于paxos、分布式系统分类，被贴了paxos、paxos member change 标签。