一致性协议概述

分布式存储系统通常通过维护多个副本来进行容错，提高系统的可用性。要实现此目标，就必须解决分布式存储系统的最核心问题：维护多个副本的一致性。

首先需要解释一下什么是一致性（consensus）,它是构建具有容错性（fault-tolerant）的分布式系统的基础。 在一个一致性的集群里面，同一时刻所有的结点对存储在其中的某个值都有相同的结果，即对其共享的存储保持一致。集群具有自动恢复的性质，当少数结点失效的时候不影响集群的正常工作，当大多数集群中的结点失效的时候，集群则会停止服务（不会返回一个错误的结果）。

常见的应用有raft的Sentinel集群。 Sentinel集群正常运行的时候每个节点epoch相同，当需要故障转移的时候会在集群中使用Raft协议选出Leader执行故障转移操作。

raft协议

集群角色 leader，follower, candidate

```
某一任期内
leader 负责发送数据
follower –接收数据
candidate–选举投票
```

选举

```
集群机器投票选出主节点–leader
```

任期(term)

```
leader 有唯一的逻辑标识其任期，参考现实生活中的领导人任期。raft中任期号严格递增
```

协议场景

Raft提供一种一致性场景，客户端调用put(x) = y, 一旦写入成功，则x的值在raft集群存在且能提供服务的情况下，一定会返回get(x) = y（但并不保证在每台raft机器上都是get(x)=y）

> 注：存在且能提供服务 <=等价于=> 集群中有法定集合存在，且法定集合数据一致

例如: 5(通常取奇数)个节点组成的server集群。客户端向集群提交 put(x) = 3,被>=3个节点确认，且在后续的请求中，没有客户端修改x的值。在存活节点>=3的情况下，客户端get(x)=3

法定集合

> 设 n个节点集群，法定集合为大于等于floor(n/2) + 1节点组成的集合

法定集合的意义：

> 假设5台机器组成的集群，发生了两次选举,根据raft协议，每次选举至少有三台机器参与才能有效。
> 则5台机器中必定有一台机器同时参与两次选举，这台参与的机器保证了数据在不同任期中的传递

核心流程

复制状态机

多台机器，如何保证状态最终一致性

> 假设每台机器由【日志序列+状态机】组成
>
> 日志顺序一致，且没有丢失 + 状态机算法固定 = 最终状态一致
>
> 即：每个节点接收到相同的操作日志序列（日志不会丢失,可能会延迟），使用相同算法执行日志对应的操作，最终状态机状态一致

例如

```
5个节点组成的机器 n1, n2, n3, n4, n5
n1 收到并执行seq['x=1', 'x=2', 'y=3', 'y=5', 'x=6'] 延时['']
n2 收到并执行seq['x=1', 'x=2', 'y=3', 'y=5']        延时['x=6']
n3 收到并执行seq['x=1', 'x=2', 'y=3']               延时['y=5','x=6']
n4 收到并执行seq['x=1', 'x=2', 'y=3', 'y=5']        延时['x=6']
n5 收到并执行seq['x=1', 'x=2', 'y=3', 'y=5', 'x=6'] 延时['']

虽然各个节点执行的序列以及x，y的值在当前时刻不一致，但是随着延时日志到达，集群所有节点最终会趋于一致
```

raft与一致状态的关系

raft 能对外输出的状态 = 法定集合最新的一致状态，在上述例子中，集群的满足raft的状态为 get(y)=5, get(x)=2

有序性的保证

任期内有序

在一个leader任期内（n1, n2, n3, n4, n5）组成集群

```
leader: n1
follower: n2, n3, n4, n5

client 提交 log(x=1) 到 n1
leader 对client log进行编号 log1(x=1), 并将log1(x=1) 推送给 follower 集合，当收到3个确认时，返回client success

client 提交 log(x=2) 到 n1
leader 对client log进行编号 log2(x=1), 并将log2(x=1) 推送给 follower 集合，当收到3个确认时，返回client success
```

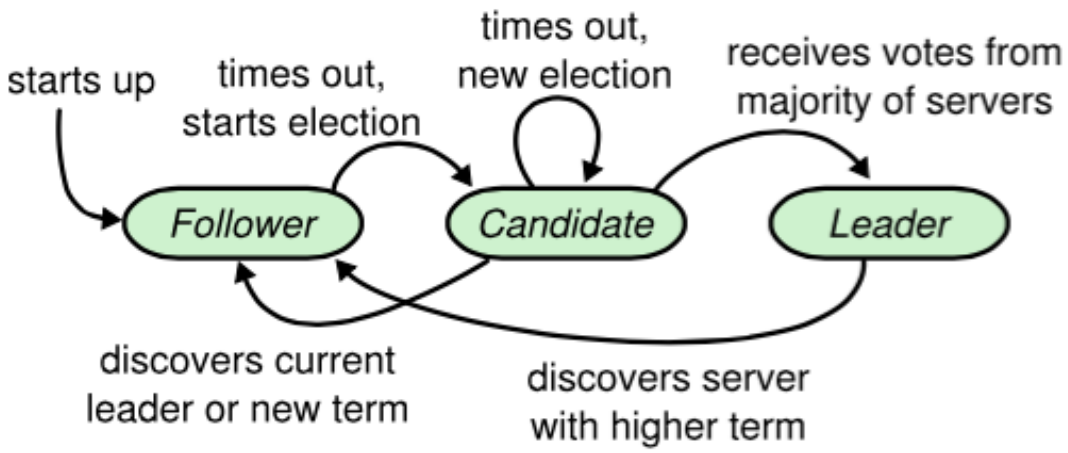
排序：此时只有一个leader对log进行排序，log列表顺序必定一致。
推送：如果follower节点没有回复确认，leader将会保留推送队列一直进行推送，最终日志必定送达。

跨任期有序

上述例子，leader宕机，或者网络隔离，将会切换leader，造成任期变更。有序被打破,需要额外的逻辑来保证全局有序。在讲解如何保证全局有序前，我们先来看下与之密切相关的选举流程

集群的角色行为及状态流转

- 流转图



- leader

```
1.初始化待同步 next index = index + 1
2.发送心跳到follower，宣布自己是master
3.接受client请求，编号，发送到follower,带上next index
4.如果某个节点回复它当前的日志 >= next index ，覆盖follower这条log
5.如果follower有日志缺失（lastIndex<nextIndex -1）,补发对应的日志
6.如果多数follower响应日志存储，apply成功，响应客户端 success
7.接受到更高term的心跳信息，切换自身状态
```

- follower

```
1.响应来自 candidate、leader rpc请求，主要包括leader的日志同步、心跳请求；
candidate的vote请求
2.follower 有自己的计时器，如果超时没收到上述任何一个合法请求 切换状态为
candidate 并发起选举
3.收到上述任何一个合法请求，重置计时器，如此往复
```

- candidate

```
1.取出当前自己获得的最大term，将term+1，投票给自己，重置计时器。
2.发送term+1的voteRequest（携带自身最新log信息）请求给集群中的机器，等待回复，以下分情况讨论：
    （1）收到大多数的投票，切换到leader，开始同步日志给follower，重置计时
    （2）收到来自其他leader的心跳或者日志请求，接受请求，修改自身log同步信息，
         切换状态为follower，重置计时
    （3）收到来自其他candidate的投票请求，比对log信息，如果对方log落后，拒绝。log信息相同，
         如果对方竞选的term更大，同意并修改自身的投票信息并重置计时，否则拒绝

注：log信息包含 log序号，log所属的term。log序号严格递增，有每个term的leader进行分配
```

日志同步

示例5个节点，格式为[term_index],例如[1_1]表示term为1，序号为1的日志。

1. 覆盖日志

```
n1 [1_1,1_2]   [1_1,1_2]    |       [1_1,2_2]
n2 [1_1,1_2]   [1_1,1_2]    |       [1_1,2_2]
n3 [1_1]       [1_1,2_2]    | ==》  [1_1,2_2]
n4 [1_1]       [1_1,2_2]    |       [1_1,2_2]
n5 [1_1]       [1_1,2_2]    |       [1_1,2_2]
```

【term-1】n1成为leader，同步日志log(1_1)给 n2,n3,n4,n5节点, n1接收到log(1_2),但是只同步到n2,n1下线
【term-2】n5成为leader(具体选举流程，白板画图)，同步log给n3,n4，n3,n4比较当前接收的log，最大term为1，最大序号为1，新同步的log满足顺序性，接收。n5同步log[2-2]给n1、n2,n1、n2发现自身已经有index=2的日志，比较term，发现term比待同步的log小，覆盖自身记录。

2.缺失

```
n1 [1_1,1_2]   [1_1,1_2]   [1_1,1_2]
n2 [1_1,1_2]   [1_1,1_2]   [1_1,1_2]
n3 [1_1]       [1_1,2_2]   [1_1,2_2,2_3] [1_1,2_2,2_3,2_4]
n4 [1_1]       [1_1,2_2]   [1_1,2_2,2_3] [1_1,2_2,2_3,2_4]
n5 [1_1]       [1_1,2_2]   [1_1,2_2,2_3] [1_1,2_2,2_3,2_4]
```

借用场景一，n1,n2节点一直没上线。n5为leader，由于集群中大部分机器存活，集群继续工作，n3,n4,n5日志序列如图。log 2_4时刻 n1,n2上线，n5 推送 [2_2,2_3,2_4] 给n1,n2 n1,n2 最终日志变成 [1_1,2_2,2_3,2_4]

全局一致性

term 总是增加的，虽然有可能有空隙，不同leader排序的日志通过term进行串接，构成全局的顺序性，根据一致性推论，所有节点的状态机算法已知，则最终所有节点必定一致。

日志压缩

leader会在appendLogRequest中带上当前已经commit的日志信息，每个node独立的对自己的系统状态进行Snapshot，并且只能对已经commit的日志记录进行snapshot，Snapshot包含快照点commit的(log index, last_included_term)，如果节点当选为leader后，append日志到follower，发现对应follower的相关日志序列已经删除，直接同步快照

动态伸缩

1.单台扩容，即每次只增删一台

当从集群中添加或删除一台机器时，任何旧配置集群中的大多数和任何新配置集群中的大多数必然存在一个交集，例如： n1,n2,n3,n4,n5 组成old集群，法定集合为3，n1,n2,n3,n4,n5,n6组成new集群，法定集合为4。新旧集群的法定集合必定有一个节点重合，此节点在参与选举的过程中只能选出一个leader，不会造成节点状态不一致。

旧集群(n1,n2,n3,n4,n5)中leader n5收到节点加入请求，n5开始同步集群配置configNew(添加一个n6节点)到follower
follower接收到 configNew 立马更新，不需要等待commit，即follower此时如果参与选举，需要4个节点才能形成法定集合
以下分成两种情况
1、如果leader顺利同步configNew到集群中的法定集合，并使用configNew规则进行提交（4个节点构成法定集合），扩容成功
2、如果leader同不configNew的过程中宕机，剩下的5台集合产生新一轮选举，已经接收configNew的按照6个节点进行，尚未接收的使用5个节点进行
    6的法定集合为4，5的法定集合为3，必有一台机器重叠，因此不可能产生双主
    情况一，oldconfig 节点赢得选举，newconfig将会被覆盖，再次提交即可
    情况二，newConfig 节点赢得选举，没问题

针对2，直接再次提交加入申请，配置总是覆盖的（画图解释）

相关实现

atomix - copycat