Raft(01)

基本概念

定义

Raft是一个共识算法，即集群中的多个节点对某某一件事能够发成一致，及时出现了部分的节点故障、网络延迟，甚至是网络分区的情况下

Raft简单来说就是一切以领导者为主，来实现一系列数据的共识，属于单领导者的分布式算法

应用

从Raft算法的定义上就可以看出，首先能解决的就是分布式数据的一致性，比较著名的就是etcd以及InfluxDB

分布式锁、分布式调取器等应用同样可由Raft算法进行实现

解决一致性以及高可用性问题

问题

在如图的网络拓扑中，假设出现了网络分区，那么Raft算法如何保证在同一时间有且只有一个领导者处理写请求的?

节点分类

既然Raft算法是单领导者的分布式算法，那么集群中的节点至少要被分成2类: 一个领导者，一批追随者。而算法本身则将节点分类成3种角色: 领导者，候选人，追随者。每个节点在同一时间内只有一个角色

成员身份

- 领导者(Leader)
  - 集群节点的领军人物，其工作是处理写请求，管理日志复制以及定时向其余节点发送心跳包
  - 心跳包的内容就是: 我还活着，你们这群小崽子们老实一点，不要随随便便推举新的领导者上位
- 候选人(Candidate)
  - 当领导者因故障下线时，候选人通过RPC调用向其它节点发送请求投票信息，通知其它节点进行投票，如果多数节点通过，将称为新的领导者
- 追随者(Follower)
  - 普通打工仔，接收领导者的一切处理需求。当领导者的心跳包发送超时时，就主动站出来推荐自己当领导者，此时身份转换成候选人

和集团的董事长以及董事会非常相似

选举领导过程

节点初始化

在集群节点初始化时，所有节点的角色均为追随者，并且，每个节点会随机地分配一个超时时间。该超时时间就是未接收到领导者心跳包的超时时间

从左图中可以看到，G1节点的超时时间最短，即最先反应过来领导者的心跳包超时了，此时G1将转变成为候选人并发起请求投票

投票阶段

每个节点在发起投票前都会给自己投一票，并且更新自己的任期编号

其它节点收到G1的请求投票消息，在任期编号为1的这届任期内，当前节点还未进行投票，并且此时G2与G3均无对应的领导者，所以将票投给节点G1，并增加自身的任期编号

最终，节点G1收到3票投票(2个节点投票，一个自身投票)，当选为这一届的领导者，并定期地向所有追随者节点发送心跳信息，防止追随者篡位

任期

领导者是有任期的，当领导者任命到期是，会重新开始选举，任期编号会随着选举过程而变化

- 当追随者接收领导人的心跳包超时时，则转变角色，由追随者转变成候选者，并将自己的任期加1

变化时机

- 当一个节点收到其它节点信息时，若发现自身的任期比对方小，则更新自己的任期
  - 例如节点G2、G3在收到任期为1的节点G1的RPC投票请求时，发现自身的任期(值为0)要比G1的任期小，则更新自己的任期
  - 若发生网络分区，某个领导者在恢复网络分区后，收到了新领导者的心跳信息时，则会立即更新任期，并且转换成追随者

关于投票

首先、每一轮的投票是有超时时间限制的，不可能永久地进行投票，该投票的超时时间为随机值。一旦超过这个值，当轮投票无效，进行下一轮投票

其次，每个节点只会进行一次投票，并且只会投给日志完整性大于或等于自身的节点

假设G5和G6同时发起投票请求，其中G5的任期编码为4，日志编码为3；G6的任期编号为3，日志编码为2；G7任期编号为3，日志编码为3。那么G7将拒绝给G6进行投票，即使G6的投票请求先于G5的投票请求到达

split vote

假设集群节点中两个节点同时发起了投票请求，并且获得的票数相同时，会发生什么?

此时G1和G2在收到投票结果并进行累加后，发现自己的票数为2，并没有多余大多数节点之和(3)，所以均放弃当选领导人，等待下一个追随者的心跳超时，并重新进行选举

日志复制

在Raft算法中，副本数据是以日志的形式存在的。当领导者接收到客户端的写操作时，处理写请求的过程就是将数据转换成日志项格式并对其进行复制

其中日志记录的并不是数据的最终状态，而是记录数据的逻辑修改，就像binlog一样

Raft日志格式

在Raft算法中，副本数据是以日志的形式存在的，而日志则由日志项所构成

日志项

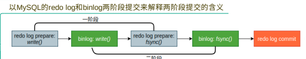
- 指令
  - 指令由客户端所制定，表示对某个key进行怎样的操作，例如 SET m=10
- 索引值
  - 当前日志项对应的索引值，为一个单调递增的整数，可以认为就是数组的索引
- 任期编号
  - 创建该条数据的领导者的任期编号

两阶段提交

Raft中日志复制非常类似于两阶段提交，只不过对其进行的优化，使得减少了一半的网络数据传输

MySQL的两阶段提交

以MySQL的redo log和binlog两阶段提交来解释两阶段提交的含义



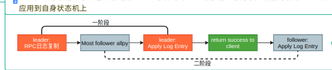
注: 图例为Group Commit优化后的提交过程

在MySQL两段提交中，最为关键的信息就是redo log所维护的redo状态: 只有在binlog已确认持久化之后才会更改redo log的状态，从而保证数据层面的一致性

Raft两阶段提交

在Raft算法中，数据的写入只能由领导者节点进行，并且在将修改同步至集群中大多数节点之前，写操作就不算真正的完成

- 当客户端写入数据时，领导者通过日志复制的RPC消息将该数据复制到追随者节点上
- 如果领导者收到了大多数追随者的确认消息，则将用户的指令应用到自身的状态机上，并返回成功给客户端
- 此后，领导者通过心跳包或者是另一个日志复制的RPC消息通知追随者，使追随者将上一条指令应用到自身状态机上



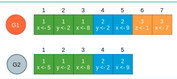
在第二阶段中，领导者并没有刻意地向追随者发送Applu Log Entry的RPC请求，而是将自身的最大索引值通过心跳包发送给追随者

由追随者自己决定是否应用未提交的日志项

一致性保证

当领导者和追随者的日志出现不一致时，领导者将强制地要求追随者直接复制自己的日志项，从而达到日志的一致性

日志覆盖过程



假设领导者G1，追随者G2的日志项如图所示，并假设索引值为7、任期为3的日志项为最新数据，将会通过RPC日志复制发送给其它追随者

领导者在发送最新数据时，还会将前一个索引项数据一起发送给追随者，最为一致性校验，在该例中，前一个日志项的索引值为6、任期为3

G2在收到RPC日志复制信息后，首先判断索引为6、任期为3的日志项是否存在，若不存在，则返回给领导者一个错误。领导者接收到该错误后，即察觉到追随者与自己已经不一致了，将会递减索引项，并将信息发送给追随者

此时领导者将发送索引项为5、任期为2的日志项给G2，G2收到数据并对比后返回成功

领导者收到成功信息后明确了G2与自己不一致的起始点，将后续所有的日志均发送给追随者，追随者直接覆盖更新