# 背景

为了解决节点损坏的问题，业界通用的方案是采用Replication，在Master写数据的时候挂多个Slave，写完后保证将log同步到Slave，这样的流程下来才能认为写入是成功的。

通常情况下为了保证性能，我们都会使用异步复制（Async Replication），不过会产生一些问题。当Master挂掉了之后很有可能有些数据并没有同步到Slave，这时候如果把这个Slave提升到新的Master就会面临数据丢失的问题。

强同步（Sync Replication）解决了异步复制（Async Replication）可能潜在的问题。Master写入数据后必须保证Slave接收到log写入才算成功，当然这样会损耗一些性能，不过最严重在于Master挂掉后，Slave提升为Master，单节点的问题就又出现了。

为了保证系统中N台节点挂掉后，仍然能够对外提供服务，至少需要2N+1台机器，也就是传统的Quorum，这样最少就需要三台机器，但是数量的增大也会带来更多的问题。

首先Master挂掉后要需要判断该提升哪个Slave为新的Master。另外在多态集群下还容易遇到一个问题，就是在网络情况下以前的Master出现网络隔离，还在继续对外提供服务，而这时新的集群和Master已经形成了，新的数据被写入到新的Master，可是读取的却是旧的数据，这就是在分布式一致性领域所面临的线性一致性问题。

分布式一致性算法最著名的应该是 Paxos，1990年提出，google的Chubby Lock服务就是使用的Paxos。之后的一些一致性算法基本都是在Paxos思路上的调整，例如ZooKeeper的ZAB。但Paxos算法一直被认为比较繁杂，很不好理解，大家对其调整优化，就是因为他的复杂。

2013年，斯坦福的两个人以易懂为目标，设计了一致性算法Raft，现在已经被广泛应用，比较有名的是etcd，Google的Kubernetes就使用了etcd作为他的服务发现框架。

**分布式一致：**

在单节点环境中，client向node发送一个值，很容易就达成一致了



但当我们有多个node时，我们应该如何做，才能实现一致性呢？

这就是分布式一致性问题，Raft就是用来解决此问题的。

# Replicated State Machine

大多数的一致性算法其实都是采用了Replicated State Machine的模型。对于这个模型你可以认为数据是被放在状态机上，用户操作集群时 首先需要写入log，然后通过一致性算法将log复制到其他机器，一旦确定其他机器都接收到log后就会认为该log被提交了，最后其他节点会依次将log存放到状态机。

## State

Raft有着三个状态，第一个状态就是Leader,即使Raft Group存在多个节点，Leader也只会存在一个，也只有Leader能负责所有数据的读写，这样就不会出现线性一致性的问题。

当一个集群选出了Leader后其他集群的节点都会变成Follow，这个Follow只负责一件事就是接收Leader的Replication logs。当一个Follow没有接收到数据或者发现集群中没有Leader的时候，就会进入Candidate状态，这个状态表示可以开始选举了。

## Term

Raft是不依赖系统的时间，它是将时间分成一个个Term，每个Term可以是任意长度，每次Term开始的时候都是由一次新的选举产生的，然后在这个Term内选出一个Leader，该Leader会一直服务到所在Leader结束。

结合Raft Term就可以进行Raft的选举。首先当系统启动后所有的节点都会进入Follow状态，Follow没有接收到任何请求的话，过一段时间就会进入Candidate状态，然后询问其他节点是否投票，如果其他节点反馈已经有新的Leader，这个Candidate就会变成Follow，而当Candidate接收到大多数的投票后就会变成Leader，之后会立即将一个log发送给其他节点开始续租Leader的时间租约。

## Log Replication

一个新的Leader被选出来之后就会开始工作了，它会不停的去接收客户端发送过来的请求，这些请求都会通过log落地，而这些log一定要是单调递增，以保证数据的一致性。

之后log会被复制到其他的节点，绝大多数节点都接收到这个log后， Leader就认为该log是committed的。

## Membership Change

对于分布式集群来说添加节点其实是分成困难的操作，最常见的做法是先更改配置中心，然后将新的配置同步到旧的节点。不过这样在同步配置的时候，就需要停止外部服务。而Raft采用了一种动态的成员节点变更，它会将新的节点到当作Raft log通过Leader传递给其他节点，这样其他节点就知道了这个新的节点的信息。不过这个过程中有可能会在某一阶段出现2个Leader的情况，为了避免这种情况就要每次只变更一个节点，而不进行多节点变更。

Raft也提供了一种多节点变更的算法，它是一种两阶段提交，Leader在第一阶段会同时将新旧集群的配置同时当成一个Raft log发送给其他旧集群节点，当这些节点接收到数据后就会和新的集群节点进入join状态，所有的操作都要进过新旧集群的大多数节点同意才会执行，然后在新的join状态内重新提交新的配置信息，在配置被committed后新的节点会上线，旧的节点会下线。

# 原理

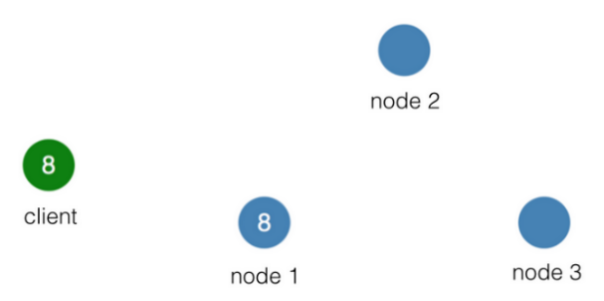
## 子问题

Paxos难以理解，raft应运而生，可以理解为简单的Paxos，它划分为三个子问题：

1. Leader Election
2. Log Replication
3. Safety

## 步骤

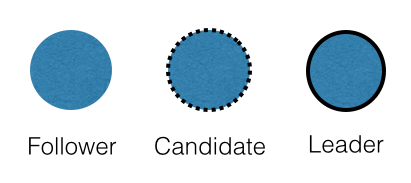
每个node都会处于以下3个状态之一：



（1）Follower 跟随者

（2）Candidate 候选人

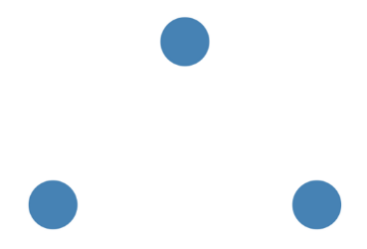
（3）Leader 领导人



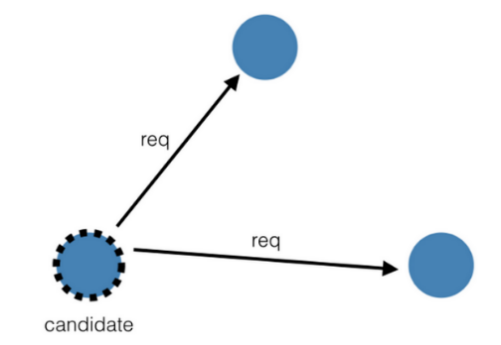
所有node开始时都是follower：

当follower没有收到leader的心跳时，他就会申请成为candidate，然后向其他node发送请求，说“我要成为Leader，请给我投票”

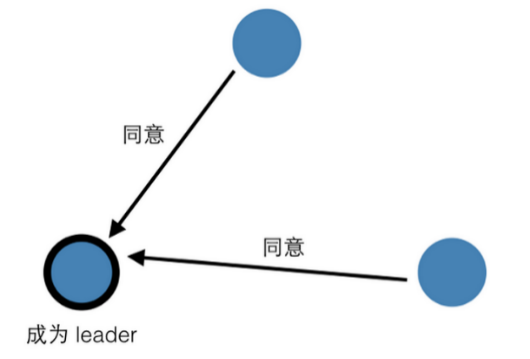
当candidate收到大多数node的同意后，就变为了Leader，以后对于系统的修改操作，都必须经过Leader

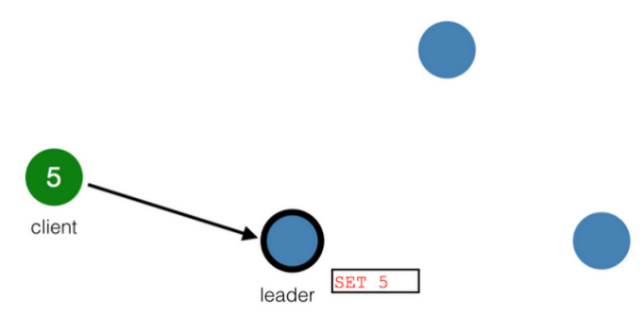


例如client要发送消息，会先发给leader，leader会把这个操作记录到自己的日志

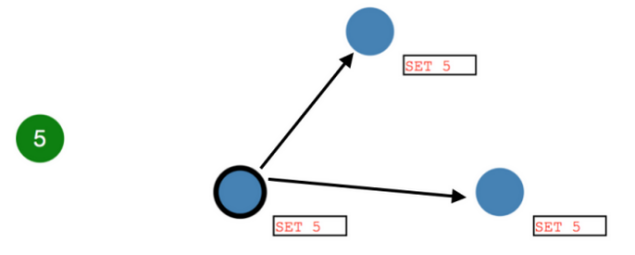


注意，是记录到日志，并没有实际修改node中的值

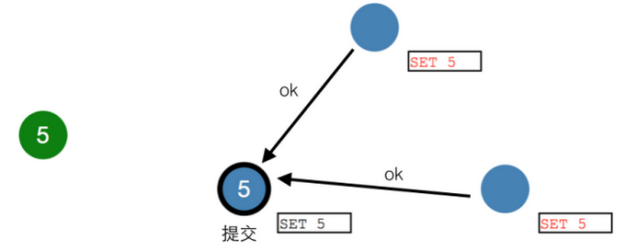




leader把这条操作记录发送给各个follower，follower收到后，也保存到自己的日志中

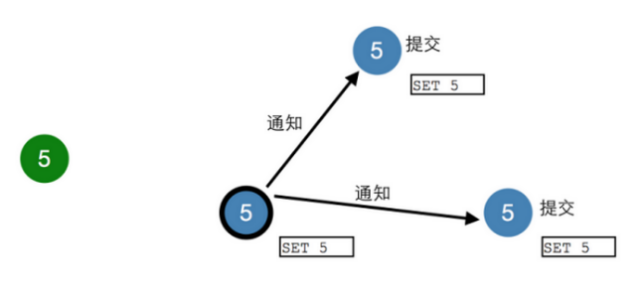


follower收到操作记录后，向leader发送消息，说自己安排好了



leader收到大多数的回馈后，就把这条记录进行提交，真正修改了node中的值

leader执行提交以后，就通知各个follower，“我已经提交了，你们可以更新了”



现在，系统就达成了一致的状态

这个过程叫做 Log Replication 日志复制，是Raft的核心之一，还有选举leader过程也是核心，就不细说了。

# 优化

## Pre-Vote

在Follow处于网络抖动无法接受到Leader的消息的时候，它就会变成Candidate并且Term加一，但是其他集群其实还是在正常工作的，这样整个集群的就混乱了。

Pre-Vote机制会在Follow没有接收到Leader的消息并且开始投票之前进入Pre-Candidate状态，在想其他节点发送投票请求，并获得同意后才会进入Candidate状态。

## Pipeline

正常的Raft流程中，客户端事先给Leader写入数据，Leader处理后会追加到log中，追加成功后Replication到其他节点中，当Leader发现log被整个集群大多数节点接收后就会进行Apply。

这样的一个过程其实是非常低效的，所以就需要引入Pipeline，它可以将多个请求进行并发处理，有效提高效率。

## Batch

通常情况下接收到的客户端请求都是依次过来的，而当有一批请求过来的时候，就可以通过Batch将这些请求打包成一个Raft log发送出去。

# Multi-Raft

当目前的Raft Group无法支撑所有的数据的时候，就需要引入Multi-Raft处理数据，第一种做法是将数据切分成多个Raft Group分担到不同的机器上。

为了应对更复杂的情况就需要使用动态分裂的方法，假设最开始的只有3台机器以及一个Region，这个Region就布置在一台机器上，所有的数据都写入这台机器，当数据量达到某个预值后Region就产生分裂，得到的新的Region就可以移植到新的机器上。

# 文档

如果对Raft算法有兴趣，强烈建议看一下他的动态演示

地址 http://thesecretlivesofdata.com/raft/

非常易懂，上面介绍的日志复制过程就是整理自这个演示，里面还有很多其他内容，看过后就会对Raft有了整体认识。

还有Raft的详细说明文档，中文的，很好的资料，地址：

https://github.com/maemual/raft-zh\_cn/blob/master/raft-zh\_cn.md

# 源码

## raft\_new

### log\_new

#### log\_alloc

##### log\_clear

#### raft\_set\_snapshot\_metadata

## ----

## raft\_recv\_requestvote\_response

### raft\_get\_nvotes\_for\_me

#### raft\_node\_has\_vote\_for\_me

### raft\_votes\_is\_majority

## raft\_recv\_requestsvote

### \_\_should\_grant\_vote

#### raft\_already\_voted

## ----

## raft\_recv\_entry

## raft\_msg\_entry\_response\_commited

## raft\_get\_first\_entry\_idx

## raft\_get\_last\_applied\_entry

## ----

## raft\_set\_election\_timeout

## raft\_set\_request\_timeout

## raft\_get\_election\_timeout

## raft\_get\_request\_timeout

## raft\_get\_timeout\_elapsed

## ----

## raft\_recv\_appendentries

### raft\_is\_candidate

### raft\_become\_follower

### raft\_delete\_entry\_from\_idx

#### log\_delete

##### raft\_node\_set\_voting

##### raft\_node\_set\_active

### raft\_append\_entry

#### raft\_entry\_is\_voting\_cfg\_change

#### log\_append\_entry

##### \_\_ensurecapacity

##### raft\_offer\_log

###### raft\_add\_non\_voting\_node

raft\_add\_node

raft\_node\_new

## raft\_recv\_appendentries\_response

### raft\_node\_get\_match\_idx

### raft\_voting\_change\_is\_progress

### raft\_node\_is\_voting\_commited

### raft\_node\_has\_suffient\_logs

## ----

## raft\_begin\_snapshot

## raft\_begin\_load\_snapshot

### log\_load\_from\_snapshot

## raft\_end\_snapshot

### raft\_get\_num\_snapshottable\_logs

#### raft\_get\_log\_count

#### log\_get\_base

### raft\_poll\_entry

#### log\_poll

### raft\_end\_load\_snapshot

### raft\_get\_snapshot\_last\_idx

### raft\_get\_snapshot\_last\_term

## ----

## raft\_periodic

### raft\_get\_num\_voting\_nodes

#### raft\_node\_is\_active

#### raft\_node\_is\_voting

### raft\_node\_is\_viting

### raft\_get\_my\_node

#### raft\_get\_nodeid

#### raft\_node\_get\_id

### raft\_is\_leader

### raft\_become\_leader

#### raft\_set\_state

#### raft\_node\_set\_next\_idx

#### raft\_get\_current\_idx

#### raft\_node\_set\_match\_idx

#### raft\_send\_appendentries

### raft\_send\_appendentries\_all

### raft\_snapshot\_is\_in\_progress

### raft\_get\_num\_voting\_nodes

### raft\_get\_commit\_idx

### raft\_apply\_all

## ----

## raft\_free

## raft\_clear

## raft\_set\_callbacks

## ----

## raft\_get\_num\_nodes

## raft\_get\_voted\_for

## raft\_get\_node\_from\_idx

## raft\_get\_current\_leader

## raft\_get\_current\_leader\_node

### log\_get\_at\_idx

## raft\_set\_last\_applied\_idx

## raft\_is\_follower

## raft\_is\_connected

## ----

## raft\_node\_get\_udata

## raft\_node\_set\_udata

## raft\_node\_has\_vote\_for\_me