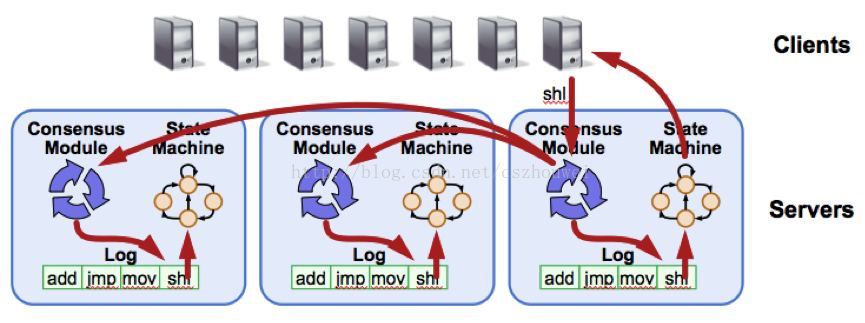
Apache Ratis

在分布式系统中节点通信存在两种模型：共享内存（Shared Memory）和消息传递（Message Passing）。一个典型场景是分布式数据库系统中，各节点初始化状态一致，每个节点都执行相同的操作序列，那么会得到一致的状态，为了保证每个节点执行相同的命令序列，需要在每一条指令上执行一致性算法以保证每个节点看到的指令一致：



其中Raft算法是常用的一致算法，其将问题抽象为ReplicatedState Machine(RSM)，如上图，每台Server保存用户命令的日志，供本地状态机顺序执行，为了保证RSM一致性，只需要保证Replicated Log的一致性即可，Server中通过Consensus Module实现每个Server上的Log一致。

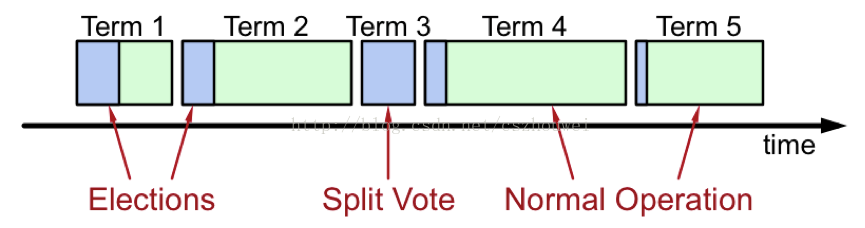
# Raft协议

Raft算法将Server划分为三种角色：Leader、Follower和Candidate，正常情况下所有的Server只有一个Leader，其他是Follower，Candiate是由Follower向Leader转换的中间状态。Server之间通过RPC消息通信，Follower不会主动发起RPC消息。Server之间的RPC有三种：

* RequestVote RPC，候选人在选举期间发起
* AppendEntries RPC，领导人发起的心跳机制，复制日志也在该命令中完成
* InstallSnapshot RPC，领导人使用该RPC来发送快照给太落后的追随者

## Leader Election

分布式环境中要保证Server的时间同步，Raft协议将时间切分为Term（逻辑时间），如下图：

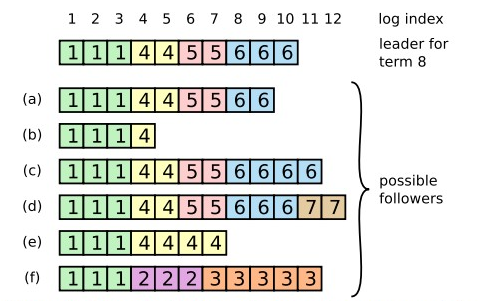


Server维护本地的currentTerm，在每个Term中至多存在一个Leader。在每个Term一开始就进行Leader Election，自增currentTerm，由Follower转换为Candidate，设置VoteFor为自身，并发起RequestVote RPC，直到满足以下任一条件：

* 获得超过半数的Server投票，转换为Leader，广播Heartbeat
* 接收到合法Leader的AppendEntries RPC，转换为Follower
* 选举超时，没有Server选举成功，自增currentTerm，重新选举

## **Log Replication**

当Leader被选举出来后，其日志可能和其他Follower日志不同，需要机制来保证日志是一致的，如下图：

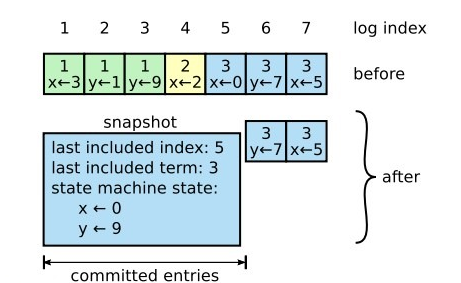


最上面的是Leader，a~f为follower，每个格子为Log Entry，其中的数字代表在哪个Term上产生。新Leader产生后，以其上的Log为准。Leader会为每个Follower维护nextIndex，表示Leader给各个Follower发送下一条Log Entry在log中的index，初始化为Leader的最后一条Log Entry的下一个位置。

完成以上操作后，Leader就可以接收客户端发来的请求了，每个请求包含一条需要被RSM执行的命令，Leader会把它作为Log Entry Append到日志中，然后给其他Server发送AppendEntries RPC。当Leader确定Log Entry被safely replicated，就apply这个Log Entry到状态及中然后返回结果给客户端，如果某个Follower宕机或者运行异常，会持续发送Rpc直到日志一致。当一条日志Commited时，Leader才决定将其apply到状态及中。

## **Log Compaction**

在实际的运行中，日志不能无限增长，否则系统重启会花费很长时间进行回放，从而影响可用性，Raft采用对整个系统进行snapshot来处理，snapshot之前的日志都可以丢弃，如下图：

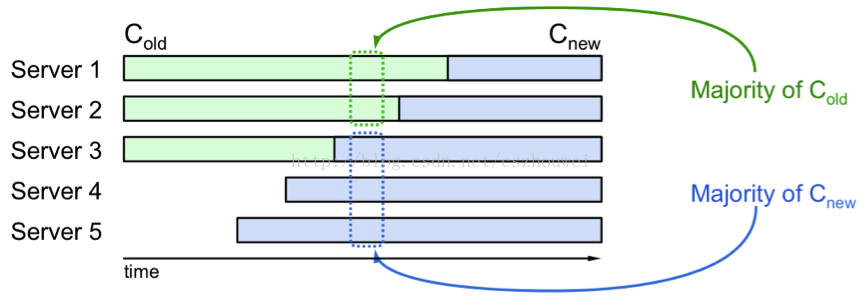


每个Server独立的对自己的系统状态进行Snapshot，并且只能对已经committed long entry进行snapshot，其中有一些元数据，包括last\_included\_index和last\_included\_term，即这条日志的termid。

Snapshot的确定是非增量的，即使内存中某个值没有变，下次Snapshot同样会被dump到磁盘。当Leader需要发给某个follower的log entry被丢弃，Leader会将Snapshot发送给落后太多的follower，当新增加一台机器时，也会发送Snapshot给它（InstalledSnapshot RPC）。

## **Cluster Membership changes**

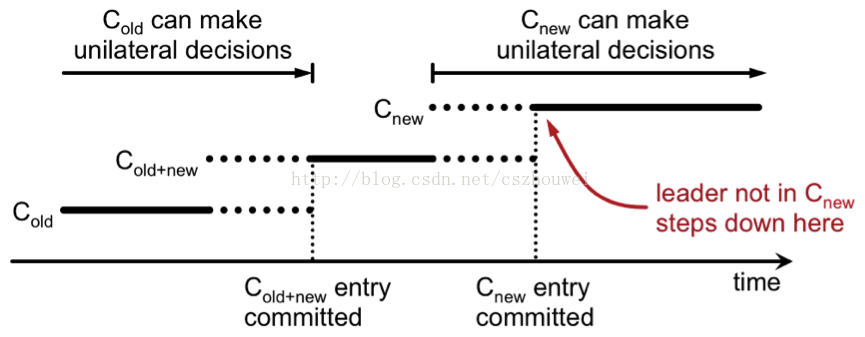
在实际系统中，由于硬件故障、负载变化等因素，机器动态增减是不可避免的，Raft采用两阶段方案：



Raft引入Join中间配置，采取两阶段方案：

* 当Leader接收到配置切换命令（Cold->Cnew），将Cold,new作为日志项进行正常的复制，任何Server一旦将新的配置项添加到本地日志，后续所有决策必须基于最新的配置项
* 当Leader确认Cold,new成功commit后，使用相同的策略提交Cnew

系统中配置切换过程如下图所示，这种方法可以解决Cold和Cnew同时生效的冲突，保证了配置切换过程的一致性，如下图：



# Apache Ratis

Apache Ratis是Raft协议的Java lib，用于保证Log副本的一致性，可以用于多Java Server之间实现状态的一致，其具有的特点：

* 支持多种RPC协议作为传输层，目前包括gRPC、Netty+Protobuf及Apache Hadoop RPC
* 支持状态机，可以使用Ratis实现自定义状态机
* 提供可插拔的Raft Log机制，用户可以实现自己的log，默认的实现是在本地文件中保存log。应用可以定义数据写入的策略

下面是示例Arithmetic示例，首先下载源代码并编译：

*$git clone <https://github.com/apache/incubator-ratis.git>*

*$mvn clean install -DskipTests*

简单的分布式计算器，将复制多实例中的数值，并执行计算，

1）总体设计

系统架构图如下：

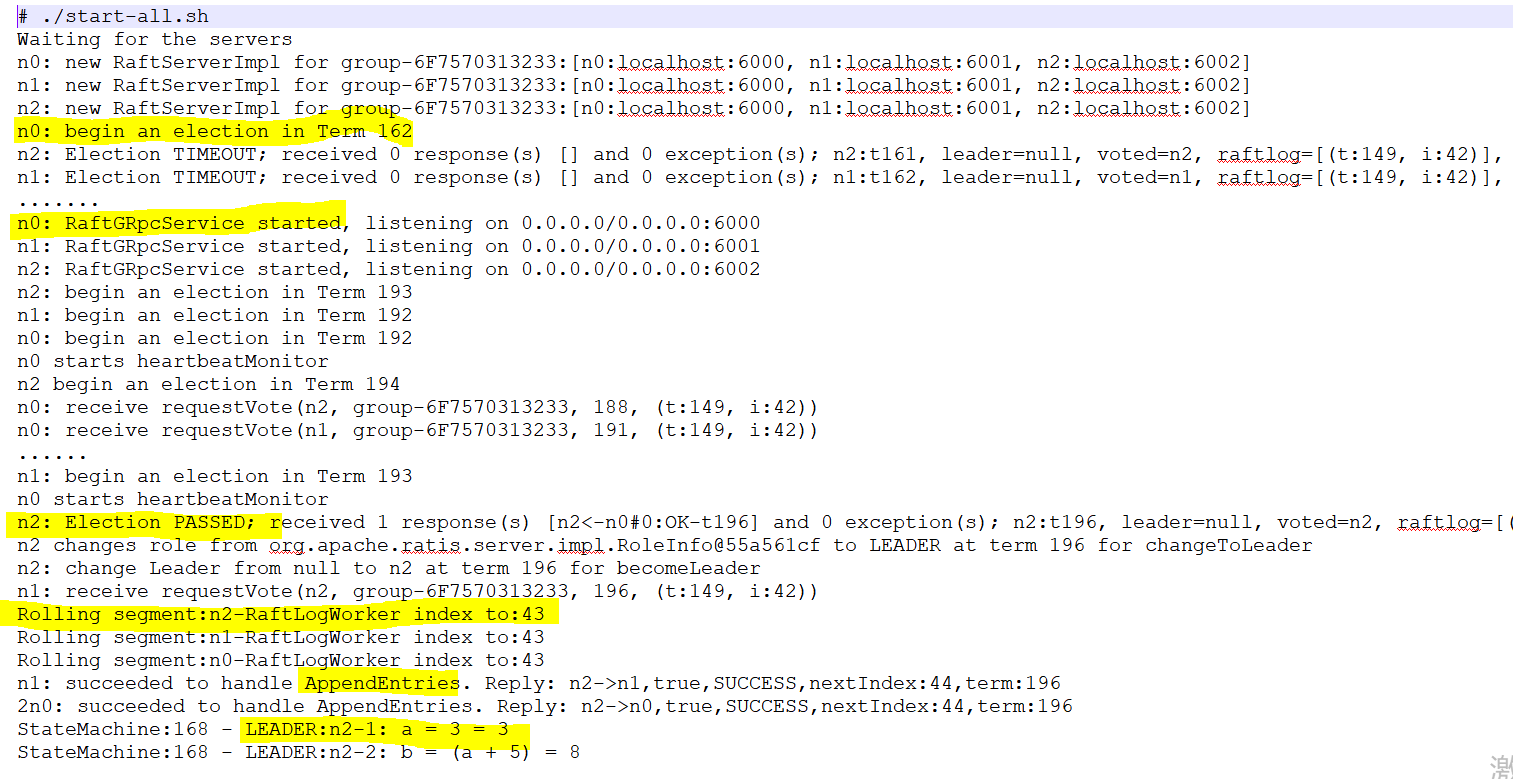


2）Server启动如下

*cd ratis-examples/src/main/bin*

*./start-all*

启动信息如下：



启动三个Server实例

*11088 ratis-examples-0.3.0-SNAPSHOT.jar*

*11089 ratis-examples-0.3.0-SNAPSHOT.jar*

*11079 ratis-examples-0.3.0-SNAPSHOT.jar*

1. 客户端

*./client.sh assign --name a --value 3*

*./client.sh assign --name b --value a+5*

*./client.sh get --name b*

输出如下：

*RaftClient:375 - client-2EA920F69C39: change Leader from n0 to n1*

*client-2EA920F69C39: receive RaftClientReply(client-2EA920F69C39->n1) in*

*group-6F7570313233, cid=0, SUCCESS, commits[n1:c44, n0:c44, n2:c44]*

*b=8*

# Apache Ratis源码分析

## **Ratis Server的启动**

启动的核心代码如下：

*//初始化PeerId*

*RaftPeerId peerId = RaftPeerId.valueOf(id);*

*//配置properties*

*/\*\**

*--peers n0:localhost:6000,n1:localhost:6001,n2:localhost:6002*

*peerId <= nx*

*address <= localhost*

*port <= 6002*

*--storageDir <= /tmp/ratis-arithmentic-$1*

*\*/*

*RaftProperties properties = new RaftProperties();*

*RaftPeer[] peers = getPeers();*

*final int port = NetUtils.createSocketAddr(getPeer(peerId).getAddress()).getPort();*

*GrpcConfigKeys.Server.setPort(properties, port);*

*properties.setInt(GrpcConfigKeys.OutputStream.RETRY\_TIMES\_KEY, Integer.MAX\_VALUE);*

*RaftServerConfigKeys.setStorageDir(properties, storageDir);*

*//初始化状态机，定义：raftGroupId*

*StateMachine stateMachine = new ArithmeticStateMachine();*

*RaftGroup raftGroup = new RaftGroup(RaftGroupId.valueOf(ByteString.copyFromUtf8(raftGroupId)), peers);*

*RaftServer raftServer = RaftServer.newBuilder()*

*.setServerId(RaftPeerId.valueOf(id))*

*.setStateMachine(stateMachine).setProperties(properties)*

*.setGroup(raftGroup)*

*.build();*

*raftServer.start();*

RaftServerProxy的启动初始代码start如下：

*public void start() {  
 JavaUtils.getAndConsume(impl, RaftServerImpl::start); //  
 getServerRpc().start();  
}*

其核心组件为RaftServerImpl及RaftServerRpc。

## 状态机

## Ratis Client的使用

https://github.com/apache/incubator-ratis

<http://ratis.apache.org/>

https://github.com/goraft/raft