本文提出了Raft共识协议，其能达到和Multi-Paxos一样的功能和性能，并且更加易懂且能更方便的应用于各种系统之中。Raft主要围绕以下Leader election、Log replication和Safety三个方面进行设计。

Raft算法中存在server和client两种角色，其中client只向server发送request，并不维护系统。因此之后提到系统中只包含server。server分为三个角色，分别是leader、follower和candidate。正常情况下，系统中只存在一个leader，其他的都是followers。follower只被动地接受leader和candidate的RPCs。server处理client的request并定时向其他所有servers发送heartbeat RPCs。candidate是在Leader election阶段临时出现的角色，稳定时要么赢得选举成为leader要么失败成为follower。

Raft算法将时间分为一个个的不定长度的term，每个term具有自己的单调递增的number。每个term最开始就是Leader election，candidate彼此竞争选举leader。若在该阶段未能选举出leader则server会发生time out事件，然后对应的server进入下一个term，再次开始Leader election。若成功则进行Log replication直至leader失效或者某follower无法收到server的heartbeat RPCs发生timeout事件。

1. Leader election

在Leader election中，candidate将自身term加一，并向其他所有server发送RequestVoteRPCs，若任一candidate收到半数以上的votes，则成为当前term的Leader。若收到其他server的AppendEntriesRPCs，且该server的term比自己的大，那么就转回follower。若在发生election timeout 事件，则开启下一轮选举。需要注意的是，任何接收到RequestVoteRPCs的follower都不会在该term中尝试成为candidate。但是这样的机制还是存在活锁的情况，即split votes的出现。Raft通过将每个server的timeout设置为在某一范围内中随机的取值来解决该问题，比如说150-300ms内取值。

1. Log replication

Leader election结束后，选举出来的Leader将向所有的server发送AppendEntriesRPCs。当Leader收到超过半数的response后就会执行该request，并称该其committed。随后其他的server将通过接收到的heartbeat RPCs判断该request是否committed，若是则执行。

若发生leader崩溃，那么follower的log将会和leader不一致。为了解决这一问题，follower在执行AppendEntriesRPCs前会执行log一致性检查。leader会查询到follower与其最后一次一致时是从哪个entry开始的，然后修改之后的所有entry直至和自己一致。这种强行修改是具有安全性的，因为Raft的设计确保了若两个server里的两个entry具有相同的index和term的话，那么在该entry之前的所有entry将保持一致的特性。

1. Safety
   1. Election restriction

为了防止后续log比较落后的server在未同步的情况下成为leader复写掉已经commit的entry。follower在接受RequestVoteRPCs时，会判断对应candidate最新的entry所在的term是否比自己大，若大则vote，若小则拒绝，若相同，则判断谁的log更长。

* 1. Committing entries from previous terms

Raft中，无法准确判断过往term中未commit的entry在之后的term中是否可以commit。因此Raft采用了比较保守的方法，即leader只commit自己所在term中的entry（因此可能会间接地提交过往term中的entry）。采用这种方法就可以简化系统，同时保证正确性。

* 1. Safety argument

Raft采用反证法证明了，任何commit之后的entry在之后的所有leader中会出现。

1. Timing and availability

一般来说，election timeout的设定要远远大于设备间通信延迟并远远小于设备从崩溃中重启的时间。

1. Cluster membership changes

当Raft系统设置发生改变时（比如说新增加两个server），之前的设计会产生不安全的状况（比如在一个term中产生两个leader）。为解决这种问题，Raft设置了一个joint consensus机制，即在旧consensus向新consensus转变的时候，必须经过joint consensu中间态，该中间态选举leader的策略是需要获取旧server一半以上的votes，同时需要新server一半以上的votes。这样的话在转变过程中，就不会出现两种leader同时存在的情况。