论文标题可能很长所以你可以把一部分放到副标题里

XXX

Raft是一种用于管理被复制的日志的算法。该算法提供与multi-Paxos等价的结果， 与Paxos一样高效，但是他的结构与Paxos不同；这使得Raft比Paxos更易于理解， 为构建切实可行的系统提供了更加坚实的基础。为了增强可理解性，Raft将共识的关键元素，比如leader选举、日志复制与安全性。 此外，Raft增强了一致性程度，并减少了需要考虑的状态数量。一项用户调查证明了对学生而言，Raft比Paxos更加易于学习。 Raft还为集群成员关系的变更引入了一种使用“重叠多数”来保证安全性的新机制。

# 绪论

共识算法可以让一组机器像一个整体一样步调一致地工作，还有能力在一些成员出现故障时保持正常。正因如此，这组机器可以在构建可靠的大规模软件系统中扮演关键角色。 在过去的几十年中，Paxos在关于共识算法的讨论中占据统治地位，大多数共识机制的实现都基于Paxos或受其影响。Paxos也成为了教授学生共识算法的首要手段。

不幸的是，尽管有很多人试图让它变得更加和蔼可亲，Paxos仍然相当难以理解。而且，它的结构需要进行一些复杂的改动才能支撑实际的系统。因此，不论是系统开发者还是学生都在与Paxos苦苦“斗争”。

在我们与Paxos“斗争”之后，我们打算找到一种新的共识算法，为系统构建与教学提供更好的基础。我们的方法有些与众不同，因为我们的主要目标是可理解性：我们能否为工程实现定义一种共识算法， 并用一种比Paxos更易于学习的方式描述它？此外我们希望这个算法能促进系统构建者的直觉的发展，这对系统开发者而言至关重要。重要的不仅是算法能工作，还有理解它的工作原理。

这些工作的成果就是一个叫做Raft的共识算法。在Raft的设计中，我们应用了特定的技术来改善颗粒接送，包括分解（Raft分离了leader选举、日志复制、和安全）和减少状态空间（相比于Paxos，Raft减少了不确定性以及服务器之间可以确保彼此的一致）。一项由来自两所大学的43名学生参与的用户调研显示，Raft的理解难度显著地低于Paxos：在学习两种算法后，其中的33名学生能更好地回答回答关于Raft的问题。

Raft在很多方面与现存的共识算法十分相似（尤其是Oki and Liskov 的 Viewstamped Replication），但是它有几个新颖的特性：

* 强leader：Raft使用了相比于其他共识算法更重要的leader角色。比如，日志条目只会从leader流向其他服务器。这简化了对日志复制的管理，使Raft更加容易被理解。
* Leader选举：Raft使用随机化的计时器来选举leader。这只在其它共识算法都需要的心跳机制基础上增加了少量内容，却让解决冲突变得更加简单快速。
* 成员关系的变更：Raft中用于改变集群中服务器集合的机制，使用了一种全新的“联合共识”的方法。在转移期间，两种配置的多数会存在重叠。这允许集群在更改配置期间能够正常运行。

# 复制状态机

共识算法通常产生于复制状态机的语境中。在这个方法中，在一系列服务器上的状态机计算出相同状态的相同拷贝；并且即使一些服务器发生故障也能正常运行。复制状态机用于解决多种分布式系统下的容错问题。例如，大规模系统拥有单独的集群leader，比如GFS、HDFS与RAMCloud，通常使用单独的复制状态机来管理leader选举与必须能从崩溃中幸存的存储配置信息。复制状态机的例子还包括Chubby和Zookeeper。

复制状态机通常用复制日志来实现，如图1所示。每个服务器都存储包含一系列命令的日志，其状态机按顺序执行该命令。每个日志都包含相同顺序的相同命令，因此每个状态机处理相同的命令序列。由于状态机是确定性的，因此每个机器计算相同的状态和相同的输出序列。

保持复制的日志一致是共识算法的任务。服务器上的共识模块从客户端接收命令并将其添加到其日志中。它与其他服务器上的共识模块通信，以确保每个日志最终包含相同顺序的同一请求，即使某些服务器失败。一旦命令被正确复制，每个服务器的状态机按日志中的顺序处理它们，并且输出返回给客户端。因此，一组服务器看起来就像一个高度可靠的单体状态机。

实际系统采用的共识算法通常由以下几个特性：

* 保证安全性（对于非拜占庭将军问题的情形：即在网络延迟、分区、丢包、重复发送、网络重排序等情况下，永不返回错误的结果）
* 只要有大多数服务器还在运行并且可以与其他服务器、客户端互相通信，系统就是完全正常运行的。因此，由五个服务器构成的集群可以容忍其中任意两台服务器的故障。假定服务器因停止而故障，他们在一段时间后会从持久化存储中恢复状态并重新加入集群。
* 不依赖时间来保证日志的一致性，有故障的始终和极端的消息延迟在最坏情况下可能导致可用性的问题。
* 一般情况下，一条命令可以在集群的大多数节点完成一轮RPC响应后完成。少数服务器的运行缓慢一般不会影响整个系统的性能。

# Paxos的问题

在过去的十年里，Leslie Lamport的Paxos几乎成为了共识的同义词。它是最常在课堂中被教授的协议，大多数共识的实现也以它为出发点。Paxos首先定义了一种能在单一决议上达成一致的协议，比如复制一个单独的日志条目。我们把这个子集称为Single-decree Paxos。Paxos随后将多个这种协议的实例联合在一起，以促进一系列的决议，比如日志（multi-Paxos）。Paxos同时保证了安全性与活跃性，同时支持集群成员的变更。它的正确性已经被证明，并且在通常情况下十分高效。

不幸的是，Paxos有两个重大缺陷。第一个缺陷是Paxos非常难以理解。它的完整阐述臭名昭著地模糊，只有很少的人在付出巨大努力后可以理解。因此，也有一些用简单的方式解释Paxos的尝试。即使这些解释专注于single-decree的子集，也同样面临很多挑战。在一个对 NSDI 2012出席者的非正式调查中，我们发现很少有人多Paxos感到舒适，即使是在富有经验的研究者人群之中。我们也曾挣扎着研究Paxos，在花费了近一年时间，阅读了好几种简化的解释并设计了我们自己自己的协议之后，我们仍然没有完全理解这个协议。我们假设Paxos的模糊性来源于他选择以single-decree的子集作为基础。Single-decree Paxos复杂而精密，他被分成两个部分却没有简单直观的解释，而且两部分不能被独立地理解。正因如此，很难建立起对single-decree协议的直观认识。Multi-Paxos的组成添加了重要的复杂性和精妙之处。我们相信对多项决议达成共识的问题（比如一个日志而不是单独的条目）可以用其他更加直接与明显的方式拆分。

Paxos的第二个问题是没有为构建实用系统提供良好的基础。其中一个原因是对于multi-Paxos而言，没有一种公认的算法。Lamport的描述大多数是关于的single-decree Paxos的。它给出了multi-Paxos的大概方法，但是缺失了很多细节。有很多充实并优化Paxos的尝试，但是它们与Lamport的描述、他们彼此之间都有区别。一些系统，比如Chubby已经实现了雷系Paxos的系统，但是大多数细节都还没有公开。

此外，Paxos的结构不适合构建真实的系统；这是以“单个决议”分解问题的另一个后果。选择一组日志的条目， 再把他们合并为一个连续的日志没有任何好处，这只能增加复杂性。围绕日志构建一个新条目只能在某种约束下 按顺序扩展的系统会更加简单和高效。另一个问题是Paxos在他的核心中使用一种对称的点对点方式（尽管这最终 被认为是为优化性能而选用的弱leader形式）。这在只需要做出一个决议的简单情况下是有道理的，但是很少有 真实的系统会采用这种方法。如果有一系列决议必须通过，更简单快速的方法是先选举出一个leader，然后让leader 来协调这些决议。

因此，实用的系统往往只有很少的部分相似。每种从Paxos出发的实现都发现了实现 它的困难之处，之后开发出了相当不同的体系。这耗时且易错，而Paxos的理解难度 加剧了这一现象。Paxos的公式也许对有利于在理论上证明它的正确性，但是真实的 实现与Paxos如此的不同，使得这些证明没有什么价值。下列的来自Chubby实现者 的评论就十分典型：

Paxos算法的描述与真实世界的系统有很大的差距....最终的系统将会基于一种未证明的的协议

由于这些问题，我们推断出Paxos没有为系统构建与教学提供良好的基础。鉴于共识 在构建大规模系统的重要性，我们决定尝试自主设计一种比Paxos更好的共识算法。 Raft就是这一尝试的产物。

# 为可理解性设计