寻找一种可理解的一致性算法（扩展版本）

作者 Diego Ongaro and John Ousterhout 斯坦福大学

摘要

Raft是一种用来管理日志复制的一致性算法。它和Paxos的性能和功能是一样的，但是它和 Paxos的结构不一样；这使得 Raft 比Paxos更容易理解并且更易于构建实际的系统。为了提高理解性，Raft 将一致性算法分为了几个部分，例如领导选取，日志复制和安全性，同时它使用了更强的一致性来减少了必须需要考虑的状态。从用户学习的结果来看， Raft 比 Paxos 更容易学会。Raft 还包括了一种新的机制来使得动态改变集群成员，它使用重叠大多数来保证安全性。

1. 引言

一致性算法允许一组机器像一个整体一样工作，即使其中的一些机器出了错误也能正常工作。正因为此，他们扮演着建立大规模可靠的软件系统的关键角色。在过去的十年中 Paxos 一直都主导着有关一致性算法的讨论：大多数一致性算法的实现都基于它或者受它影响，并且 Paxos 也成为了教学生关于一致性知识的主要工具。

不幸的是，尽管在降低它的复杂性方面做了许多努力，Paxos 依旧很难理解。并且，Paxos 需要经过复杂的修改才能应用于实际中。这些导致了系统构构建者和学生都十分头疼。

在 被 Paxos 折磨之后，我们开始寻找一种在系统构建和教学上更好的新的一致性算法。我们的首要目标是让它易于理解：我们能不能定义一种面向实际系统的一致性算法并且比 Paxos 更容易学习呢？并且，我们希望这种算法能凭直觉就能明白，这对于一个系统构建者来说是十分必要的。对于一个算法，不仅仅是让它工作起来很重要，知道它是如何工作的更重要。

我们工作的结果是一种新的一致性算法，叫做 Raft。在设计 Raft 的过程中我们应用了许多专门的技巧来提升理解性，包括算法分解（分为领导选取（leader selection），日志复制（log replication）和安全性（safety））和减少状态（state space reduction）（相对于 Paxos，Raft 减少了非确定性的程度和服务器互相不一致的方式）。在两所学校的43个学生的研究中发现，Raft 比 Paxos 要更容易理解：在学习了两种算法之后，其中的33个学生回答 Raft 的问题要比回答 Paxos 的问题要好。

Raft 算法和现在一些已经有的算法在一些地方很相似（主要是 [Oki 和 Liskov 的 Viewstamped Replication](http://www.pmg.csail.mit.edu/papers/vr.pdf)。但是 Raft 有几个新的特性：

* 强领导者（Strong Leader）：Raft 使用一种比其他算法更强的领导形式。例如，日志条目只从领导者发送向其他服务器。这样就简化了对日志复制的管理，使得 Raft 更易于理解。
* 领导选取（Leader Selection）：Raft 使用随机定时器来选取领导者。这种方式仅仅是在所有算法都需要实现的心跳机制上增加了一点变化，它使得在解决冲突时更简单和快速。
* 成员变化（Membership Change）：Raft 为了调整集群中成员关系使用了新的联合一致性（joint consensus）的方法，这种方法中大多数不同配置的机器在转换关系的时候会交迭（overlap）。这使得在配置改变的时候，集群能够继续操作。

我们认为，Raft 在教学方面和实际实现方面比 Paxos 和其他算法更出众。它比其他算法更简单、更容易理解；它能满足一个实际系统的需求；它拥有许多开源的实现并且被许多公司所使用；它的安全特性已经被证明；并且它的效率和其他算法相比也具有竞争力。

这篇论文剩下的部分会讲如下内容：复制状态机（replicated state machine）问题（第2节），讨论 Paxos 的优缺点（第3节），讨论我们用的为了达到提升理解性的方法（第4节），陈述 Raft 一致性算法（第5~8节），评价 Raft 算法（第9节），对相关工作的讨论（第10节）。

2. 复制状态机（Replicated State Machine）

一致性算法是在[复制状态机](https://www.cs.cornell.edu/fbs/publications/SMSurvey.pdf)的背景下提出来的。在这个方法中，在一组服务器的状态机产生同样的状态的副本因此即使有一些服务器崩溃了这组服务器也还能继续执行。复制状态机在分布式系统中被用于解决许多有关容错的问题。例如，GFS，HDFS还有 RAMCloud 这些大规模的系统都是用一个单独的集群领导者，使用一个单独的复制状态机来进行领导选取和存储配置信息来应对领导者的崩溃。使用复制状态机的例子有 Chubby 和 ZooKeeper。

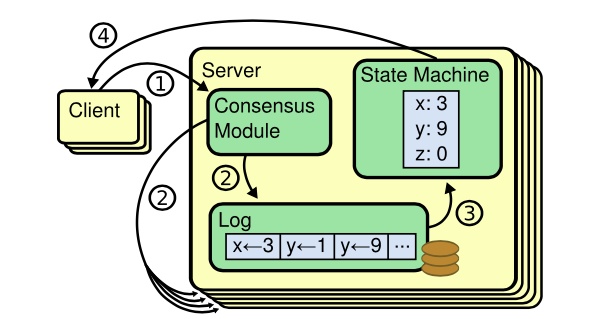


图-1：复制状态机的架构。一致性算法管理来自客户端状态命令的复制日志。状态机处理的日志中的命令的顺序都是一致的，因此会得到相同的执行结果。

如图-1所示，复制状态机是通过复制日志来实现的。每一台服务器保存着一份日志，日志中包含一系列的命令，状态机会按顺序执行这些命令。因为每一台计算机的状态机都是确定的，所以每个状态机的状态都是相同的，执行的命令是相同的，最后的执行结果也就是一样的了。

如何保证复制日志一致就是一致性算法的工作了。在一台服务器上，一致性模块接受客户端的命令并且把命令加入到它的日志中。它和其他服务器上的一致性模块进行通信来确保每一个日志最终包含相同序列的请求，即使有一些服务器宕机了。一旦这些命令被正确的复制了，每一个服务器的状态机都会按同样的顺序去执行它们，然后将结果返回给客户端。最终，这些服务器看起来就像一台可靠的状态机。

应用于实际系统的一致性算法一般有以下特性：

* 确保安全性（从来不会返回一个错误的结果），即使在所有的非拜占庭（Non-Byzantine）情况下，包括网络延迟、分区、丢包、冗余和乱序的情况下。
* 高可用性，只要集群中的大部分机器都能运行，可以互相通信并且可以和客户端通信，这个集群就可用。因此，一般来说，一个拥有 5 台机器的集群可以容忍其中的 2 台的失败（fail）。服务器停止工作了我们就认为它失败（fail）了，没准一会当它们拥有稳定的存储时就能从中恢复过来，重新加入到集群中。
* 不依赖时序保证一致性，时钟错误和极端情况下的消息延迟在最坏的情况下才会引起可用性问题。
* 通常情况下，一条命令能够尽可能快的在大多数节点对一轮远程调用作出相应时完成，一少部分慢的机器不会影响系统的整体性能。

3. Paxos 算法的不足

在过去的10年中，Leslie Lamport 的 Paxos 算法几乎已经成为了一致性算法的代名词：它是授课中最常见的算法，同时也是许多一致性算法实现的起点。Paxos 首先定义了一个能够达成单一决策一致的协议，例如一个单一复制日志条目（single replicated log entry）。我们把这个子集叫做单一决策 Paxos（single-decree Paxos）。之后 Paxos通过组合多个这种协议来完成一系列的决策，例如一个日志（multi-Paxos）。Paxos 确保安全性和活跃性（liveness），并且它支持集群成员的变更。它的正确性已经被证明，通常情况下也很高效。

不幸的是，Paxos 有两个致命的缺点。第一个是 Paxos 太难以理解。它的完整的解释晦涩难懂；很少有人能完全理解，只有少数人成功的读懂了它。并且大家做了许多努力来用一些简单的术语来描述它。尽管这些解释都关注于单一决策子集问题，但仍具有挑战性。在 NSDI 2012 会议上的一次非正式调查显示，我们发现大家对 Paxos 都感到不满意，其中甚至包括一些有经验的研究员。我们自己也曾深陷其中，我们在读过几篇简化它的文章并且设计了我们自己的算法之后才完全理解了 Paxos，而整个过程花费了将近一年的时间。

我们假定 Paxos 的晦涩来源于它将单决策子集作为它的基础。单决策（Single-decree）Paxos 是晦涩且微妙的：它被划分为两个没有简单直观解释的阶段，并且难以独立理解。正因为如此，它不能很直观的让我们知道为什么单一决策协议能够工作。为多决策 Paxos 设计的规则又添加了额外的复杂性和精巧性。我们相信多决策问题能够分解为其它更直观的方式。

Paxos 的第二个缺点是它难以在实际环境中实现。其中一个原因是，对于多决策 Paxos （multi-Paxos） ，大家还没有一个一致同意的算法。Lamport 的描述大部分都是有关于单决策 Paxos （single-decree Paxos）；他仅仅描述了实现多决策的可能的方法，缺少许多细节。有许多实现 Paxos 和优化 Paxos 的尝试，但是他们都和 Lamport 的描述有些出入。例如，Chubby 实现的是一个类似 Paxos 的算法，但是在许多情况下的细节没有公开。

另外，Paxos 的结构也是不容易在一个实际系统中进行实现的，这是单决策问题分解带来的又一个问题。例如，从许多日志条目中选出条目然后把它们融合到一个序列化的日志中并没有带来什么好处，它仅仅增加了复杂性。围绕着日志来设计一个系统是更简单、更高效的：新日志按照严格的顺序添加到日志中去。另一个问题是，Paxos 使用对等的点对点的实现作为它的核心（尽管它最终提出了一种弱领导者的形式来优化性能）。这种方法在只有一个决策被制定的情况下才显得有效，但是很少有现实中的系统使用它。如果要做许多的决策，选择一个领导人，由领带人来协调是更简单有效的方法。

因此，在实际的系统应用中和 Paxos 算法都相差很大。所有开始于 Paxos 的实现都会遇到很多问题，然后由此衍生出了许多与 Paxos 有很大不同的架构。这是既费时又容易出错的，并且理解 Paxos 的难度又非常大。Paxos 算法在它正确性的理论证明上是很好的，但是在实现上的价值就远远不足了。来自 Chubby 的实现的一条评论就能够说明：

Paxos 算法的描述与实际实现之间存在巨大的鸿沟…最终的系统往往建立在一个没有被证明的算法之上。

正因为存在这些问题，我们认为 Paxos 不仅对于系统的构建者来说不友好，同时也不利于教学。鉴于一致性算法对于大规模软件系统的重要性，我们决定试着来设计一种另外的比 Paxos 更好的一致性算法。Raft 就是这样的一个算法。

## 4. 易于理解的设计

设计 Raft 的目标有如下几个：

* 它必须提供一个完整的、实际的基础来进行系统构建，为的是减少开发者的工作；
* 它必须在所有情况下都能保证安全可用；
* 它对于常规操作必须高效；
* 最重要的目标是：**易于理解**，它必须使得大多数人能够很容易的理解；
* 另外，它必须能让开发者有一个直观的认识，这样才能使系统构建者们去对它进行扩展。