论文信息

本次报告主要围绕一致性算法展开，介绍两种受欢迎的一致性算法——Paxos 和 Raft，并将这两种“风格迥异”的一致性算法进行对比。参考的论文主要包括：

- Lamport, L. *Paxos made simple*. ACM SIGACT News 32, 4 (Dec. 2001), 18–25
- Diego Ongaro and John Ousterhout, Stanford University. *In Search of an Understandable Consensus Algorithm*. USENIX 2014 (best paper)

本次报告的结构如下：

第一章：介绍分布式系统中的一致性问题，阐述一致性算法要解决的问题。

第二章：详细介绍一致性算法的鼻祖——Paxos，包括个人角度理解的 Paxos 一致性保证的讨论。

第三章：详细介绍更“年轻”、更容易理解的 Raft 一致性算法，主要包括 Leader selection 和 Log replication 过程，以及其日志一致性的讨论。

第四章：对比 Raft 相比于 Paxos 的异同。

1 一致性问题

一致性算法用来解决一致性问题，而一致性问题是分布式系统中的经典问题。

为了解决单机操作系统计算及存储资源的不足，分布式系统应运而生。为了避免单点故障，数据在分布式系统中通常有多个备份，这使得分布式集群有更好的错误容忍度。一致性问题是指对于一组服务器，给定一组操作，我们需要一个协议（一致性算法）使得最后它们的结果达成一致。更详细地说，当其中某个服务器收到客户端的一组指令时，它必须与其它服务器交流以保证所有的服务器都是以同样的顺序收到同样的指令，这样，所有的服务器会产生一致的结果，看起来就像是一台机器一样。

2 Paxos

Paxos 算法是 Lamport 于 1990 年提出的一种基于消息传递（不考虑 Byzantine 事件）且具有高度容错特性的一致性算法。

起初，Lamport 用 Greek 的例子来描述这个算法的主要思想，但实在难以理解；之后，Lamport 用英文描述了 Paxos 算法，虽然可读性提高，但是 Paxos 算法本身还是十分复杂，需要阅读更多的资料才能够基本理解 Paxos。这也正是之后的一些更易理解的一致性算法出现的初衷。

Paxos 算法解决的问题是一个分布式系统如何就某个值 (决议) 达成一致。它实际上是一个共识算法。

2.1 问题描述

设想一组可以提出提议的 process。一致性算法保证所有提出的提议中，只有一个提议会被选择。如果没有提出提议，那么将不会有提议被选择。如果一个提议被选择，那么所有 process 可以学习到这个提议。一致性算法需要的安全性要求是:

1. 仅可以选择被提出的提议
2. 仅有一个提议会被选择
3. process 不会知晓一个值被选择了，直到这个值确实被选择了

2.2 Paxos 一致性算法

Paxos 算法中划分了 3 个角色：proposer（提出者）、acceptor（批准者）和 learner（学习者）。一个 process 可以扮演多个角色，并假设 process 之间的通信是基于消息的、异步的、非 Byzantine 模型的。

每个提议有一个为实数的编号 n 以及提议内容 v，因为一个提议用 {n, v} 表示，其中 n 全序且唯一的。

2.2.1 达成一致的过程

两个阶段 选择唯一的提议的过程分为两个阶段：

- 阶段一：Prepare
  1. Proposer：选择一个提议，编号为 n，向 acceptor 的多数派发送编号为 n 的 prepare 请求。
  2. Acceptor：如果接收到的 prepare 请求的编号 n 大于它已经回应的任何 prepare 请求，它就回应此请求。Acceptor 将告知已经接受的编号最高的提议的内容，并承诺不再回应任何编号小于 n 的提议。
- 阶段二：Accept
  1. Proposer：如果收到了多数 acceptor 对 prepare 请求（编号为 n）的回应，它就向这些 acceptor 发送提议 {n, v} 的 accept 请求，其中 v 是所有 acceptor 回应中编号最高的提议的内容（如果收到的所有 acceptor 的回应都不带有其已经接受的提议，说明当前提议将是这些 acceptor 收到的第一个提议，这时，提议的内容可以是任意的）。
  2. Acceptor：当收到提议 {n, v}，如果此 acceptor 在此期间没有接受过任何编号大于 n 的提案，则接受此提案内容，否则忽略。

一致性保证 在 Paxos 论文中，包含关于以上算法的一致性的论证。以下就我个人角度及观点，讨论以下两个 Paxos 的一致性要求：

1. Acceptor 必须接受它接收到的第一个提议。
2. 如果一个提议 {n, v} 被选择/接受，那么每一个编号更高的被选择的提议包含的值都是 v。

第一个要求是显而易见的，在没有消息丢失等异常的情况下，即使仅有一个 proposer 提出了一个提议，我们也希望 acceptor 能够接受这个提议。因此，acceptor 应该接受它接收到的第一个提议（对应到 Paxos 的 Accept 阶段，acceptor 接受收到的第一个 Accept 请求）。

Paxos 的 Prepare 阶段保证了第二个要求，它使得：如果一个提议 {n, v} 被选择，那么每一个 acceptor 接受的编号更高的提议包含的值都是 v。这是通过 acceptor 在 prepare 阶段对 proposer 的回

应保证的，prepare 阶段能够保证：如果一个提议 {n, v} 已经被返回 prepare 回应的 acceptor 选择，那么此后，任何 proposer 提出的编号更高提议包含的值都是 v。

结合这两个要求，我们可以通过简单的类似第二数学归纳法的思想来证明一致性。就我个人的理解，在这个问题中，要求一找到了第一个被接受的提议，相当于保证了"最小数"依据的存在；要求二保证了后续的提议相对于之前的提议来说都是可归纳的。从而 Paxos 能够保证一致性过程结束后能够达到最终一致性。

2.2.2 处理流程

Learner 获知已经被选择的值 达成一致性的过程是 proposer 和 acceptor 两个角色之间的交互,learner 角色需要在达到一致之后获得一致的结果。

最简单的方法是，让每个 acceptor 在批准提议的时候通知所有的 learner，把批准的提议发给 learner。但一个显而易见的问题是，告知 learner 所需的消息数量（learner 数与 acceptor 数的乘积）过于庞大。

另一种方法是，让 acceptor 将接受情况回应给一个主 learner，它再把被选择的值通知给其它的 learner。这增加了一次额外的消息传递，会比前一种不可靠一些，因为主 learner 可能会失效，但是这个方法要求的消息个数仅是 learner 数和 acceptor 数的总和。

更一般的，acceptors 可以通知多个主 learners，每个主 learner 都能通知其它所有的 learners。主 learner 越多越可靠，但是通信复杂性会增加。

为了确保达到一致过程能够结束，引入主 proposer Paxos 选择唯一的提议的过程确实能够保证一致性，但这个达到最终一致性的过程对应到实际的处理流程时依旧存在问题。

设想这样一个场景，两个 proposer 轮流提出一系列编号递增的议案，但是都没有被选择。Proposer p 用提议的编号为 n1，并结束阶段 1; 另外一个 proposer q 选择了提议编号 n2 > n1，并结束阶段 1。于是 p 在阶段 2 的 accept 请求将被忽略，因为 acceptor 已经承诺不再批准编号小于 n2 的议案。于是 p 再回到阶段 1 并选择了编号 n3 > n2，这又导致 q 第二阶段的 accept 请求被忽略; 这样一直循环下去……

为了避免这种情况（即保证达到最终一致的过程能够结束），Paxos 规定必须选择一个主 proposer——只有主 proposer 才能提出提议。如果主 proposer 和多数 acceptor 成功通信，并提出一个编号更高的提议，提议将被接受，值会被选择。如果它得知已经有编号更高的议案，它将放弃当前的提议，主 proposer 最终能选择一个足够大的编号。

Paxos 并未对主 proposer 的选择做规定，但 Paxos 指出选择主 proposer 的可靠算法必须使用随机或者实时——例如，使用超时机制。

2.2.3 实现

Paxos 算法假设了一个某些 process 组成的网络。在其一致性算法中，每个 process 都同时扮演 proposer、acceptor 和 learner 的角色。算法选择一个 leader，它就是主 proposer 和主 learner。在考虑实现时，需要使用持久化存储来保证 acceptor 失效后也能记住必要的信息，acceptor 在发送响应前必须持久化存储该响应。

此外，需要一个保证任何两个提议的编号都不相同的机制。Proposer 从互不相交的集合中选择议案编号，因此两个不同的 proposer 永远不会提出相同编号的议案。每个 proposer 都持久化保存它已经提出的编号最高的提议，并使用一个更高的提议编号来开始 Prepare 阶段。

3 Raft

Paxos 是分布式系统中最早的一致性算法，它与“更年轻的”Raft 算法都是目前最受欢迎的一致性算法。相比于 Paxos，Raft 更加容易理解。在 Raft 中，server 之间是基于消息（RPC）通信的。与 Paxos 不同，它将一致性的过程分为 Leader Election 和 Log Replication 两个部分。

服务器状态 在任何一个时刻，Raft 集群中的 server 处于 Leader、Follower 或者 Candidate 三种状态之一。Candidate 状态的 server 只会出现在 Leader election 阶段。在正常操作过程只有一个 server 作为 Leader，其他 server 作为 Follower 被动地处理来自 Candidate 和 Leader 的请求。图 1是三个状态之间的转换，我将在后面的章节中详细描述。

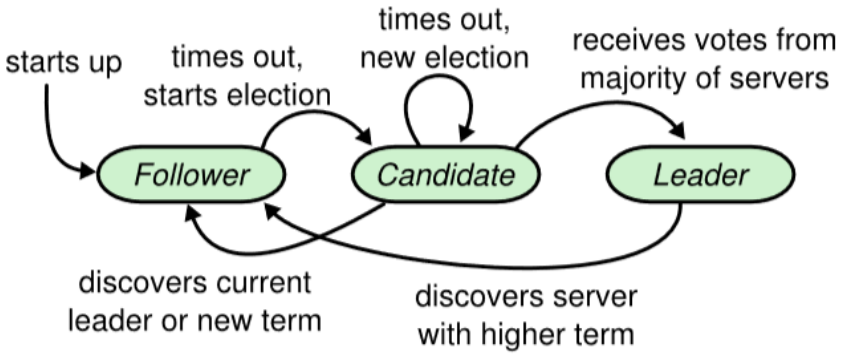

图 1: Raft server 状态转化

任期 Raft 将时间分为多个连续的任期 term，如图 2所示，每个任期从 Leader election 阶段开始。Leader election 结束后选举出唯一的 server 作为这个任期的 Leader，Leader 周期性地向所有 Follower 发送 heartbeat 来宣告自己能够胜任 Leader；Leader election 也可能会出现选票瓜分从而无法选出唯一的 Leader 的情况，这时直接结束这个任期，开始下个任期。因此，Raft 确保了每个任期最多只有一个 Leader。

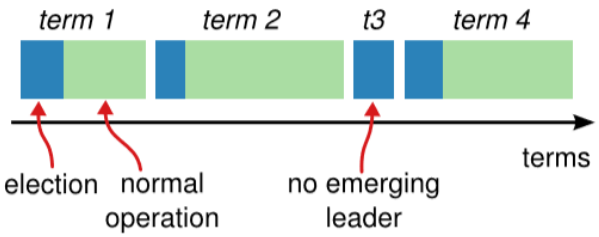

图 2: Raft 任期

3.1 Leader Election

Learn Election 过程用来选出最多一个 server 作为任期的 Leader。

3.1.1 具体过程

Raft 通过 heartbeat 机制来触发 Leader election。初始时，所有 server 都是 Follower 状态。

Follower

- 回应来自 Candidate 和 Leader 的请求。
- 如果 Follower 在 election timeout 内没有收到 heartbeat 消息，它会认为 Leader 这时候可能发生了故障，从而开始一个 Leader election 阶段来选出新的 Leader。注意，election timeout 取 150-300ms 之间的随机值，从而极大程度地减少了选票瓜分情况发生的概率。

  在 Leader election 开始时，Follower 更新任期并转变为 Candidate。

Candidate

- 转变为 Candidate 后:

  更新任期

  重置 election timer

  向集群中的其他 server 分发选票

- 如果收到了大多数 server 的选票，则胜出，成为这个任期的 Leader。
- 如果收到了 heartbeat 消息:

  heartbeat 消息中夹带的任期信息没有过期，则已经选出 Leader，它将转变为 Follower。

  heartbeat 消息中夹带的任期信息已过期，继续 Candidate 状态。

- 如果在一段时间内没有出现以上两种情况，说明可能有多个 Follower 同时转变为 Candidate 状态，并将选票瓜分了，从而没有一个 Candidate 能够收到足够的选票从而成为 Leader。此时，Candidate 触发超时，结束这个任期，开始新一任期的 Leader election。

Leader 从 Candidate 转变为 Leader 后，它向其他所有 server 周期性地发送 heartbeat 消息来宣告自己是这个任期内的 Leader 并 ni 正常运作（防止 Follower 触发 election timeout）。

3.2 Log Replication

3.2.1 日志组织结构

Raft 的日志组织结构如图 3所示。每个日志条目存储一条状态机指令和领导人收到该指令时的任期号（任期号用于检测多个日志副本之间不一致的情况）。为了保证日志匹配的原则，每个条目都有一个整数索引来表示它在日志中的位置。

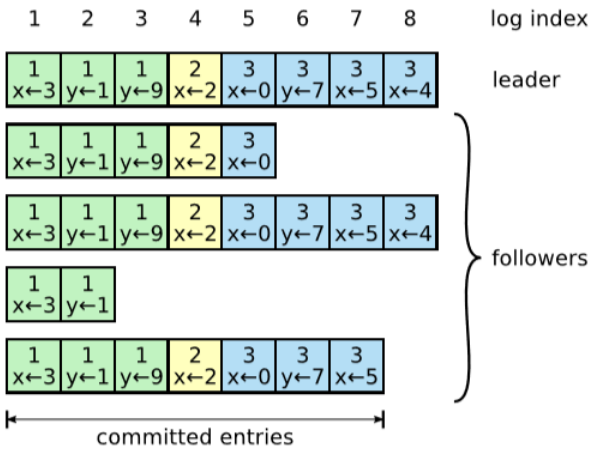

图 3: Raft 的日志结构

3.2.2 具体过程

Leader 在 Leader Election 成功选出 Leader 之后，Leader 就开始接收 Client 请求。

- Leader 收到 Client 的请求，生成对应的日志条目，并把生成日志条目的请求并行地广播给所有的 Follower。
- 如果超过一半的 Follower 都返回 success，Leader 会认为这个条目是可提交的，Leader 就会正式执行这个日志条目，并将执行结果返回给 Client。

Follower 每个 Follower 在收到 Leader 的请求后有两种选择：

- 听从 Leader 的请求，写入日志条目并返回 success。
- 在某些条件不满足的情况下，Follower 无法写入日志条目，返回 false。

另外，在 Raft 算法中，Leader 通过强制 Follower 复制它的日志来处理日志的不一致。这就意味着，在 Follower 上的冲突日志会被 Leader 的日志覆盖掉。

3.2.3 日志的一致性讨论

Raft 中日志的一致性主要基于 Raft 的日志机制能够满足以下两个特性：

- 如果在不同的日志中的两个条目拥有相同的索引和任期号，那么他们存储了相同的指令。这是因为，Leader 在一个任期里在给定的日志索引位置最多创建 1 条日志条目，同时该条目在日志中的位置也不会改变。
- 如果在不同的日志中的两个条目拥有相同的索引和任期号，那么他们之前所有的日志条目也全部相同。这个特性源于 heartbeat 的一个简单的一致性检查。当发送一个 heartbeat 时，Leader 会把新的日志条目紧接着的之前的条目的索引位置和任期号都包含在里面。如果 Follower 没有在它的日志中找到相同索引和任期号的日志，它就会拒绝新的日志条目。这个一致性检查就像一个归纳的步骤，只要 heartbeat 返回成功的时候，Leader 就知道 Follower 的日志和它的是一致的。

4 对比 Raft 与 Paxos

通过以上对 Paxos 和 Raft 的描述，应该能够很直观地感受到 Paxos 一致性算法本身是有些复杂并且难以推理的，而 Raft 算法十分清晰，并且易于实施。因此，研究人员通常对 Paxos 更感兴趣，但 Raft 更受到工程师的欢迎。Raft 与 Paxos 的"年纪"相差二十年有余，Raft 在为一致性算法注入新鲜血液的同时也与 Paxos 算法有一些相同之处。

服务器——确定状态机 两者在涉及到实现时，都将 server 看作是确定状态机（Raft 沿用了 Paxos），一致性过程保证了每台状态机执行相同的指令从而得到最终的一致性结果。

唯一 Leader Paxos 和 Raft 在正常操作期间都有唯一一个 Leader 来协调系统的一致性。唯一的 Leader 协调系统中所有 server 的状态，使得一致性算法更容易规定和实施。同时，这个唯一的 Leader 也可能成为系统的性能瓶颈。

Leader Election Raft 中明确规定了 Leader Election 过程，在这个过程中也暗示了 Raft 中只有拥有最新信息的 server 才会成为 Leader；而 Paxos 中没有定义如何推举出 Leader，任何一个 server 都可能成为 Leader。

决策方案 Paxos 和 Raft 所遵循的决策方案本质上是相同的——多数同意即可（Raft 沿用了 Paxos）。

活跃度 两种算法的决策方案决定了它们都可以保证：只要大多数 server 还正常工作，就能正常提供服务。