Raft 算法的领导者选举实验

李青坪[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)
通讯作者: 李青坪, E-mail: lqp19940918@163.com

摘 要: 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 也成为了教学中关于一致性知识的主要工具。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。Raft 的设计过程应用了许多专门的技巧来提升理解性，使得 Raft 更容易理解并且更易于建立实际的系统。本文实现了 Raft 算法中领导者选举的步骤。

关键词: 一致性算法;Paxos 算法;Raft 算法;领导者选举

Distributed File System Experiment Based on RPC

LI Qingping[1]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)

Abstract: Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems.In the past ten years, Paxos has always dominated the discussion of consensus algorithms, and most consensus algorithms are implemented based on it or affected by it, and Paxos has also become a major tool in teaching about coherence. Raft is a consensus algorithm for managing log-replication. It behaves the same as Paxos in performance and functionality, but it does not have the same structure as Paxos. Raft's design process uses a number of specialized techniques to improve comprehension, making Raft easier to understand and easier to build on actual systems. This paper implements the step of leader election.

Key words: consensus algorithm; Paxos algorithm; Raft algorithm; leader election

1 Raft 简介

在 Raft 被提出来之前，Paxos 协议是第一个被证明的一致性算法，但是 Paxos 的论文非常难懂，导致基于 Paxos 的工程实践和教学都十分艰难。于是 Raft 在设计的过程中，就从可理解性出发，使用包括算法分解（分为领导选取（leader selection），日志复制（log replication），安全性（safety））和减少状态（state space reduction）的方式增强了一致性算法的可理解性。

2 系统设计与实现

2.1 系统设计要求

实现 Raft 领导者选举。具体要求如下：

通过往 raft.go 中添加代码实现 Raft。

首先需要补齐 RequestVoteArgs 和 RequestVoteReply 结构。

修改 make()方法，当一段时间没有收到另一个结点的心跳信息后，创建一个后台 go routine，通过发送 RequestVote RPC 来开始一轮选举。

定义 AppendEntries 结构，用来发送心跳。

2.2 系统实现

2.2.1 整体实现过程

分布式系统中的所有结点都是平等的，当结点处于 follower 状态时，每个结点等待 leader 结点发来的心跳，维持 leader-follower 关系；当某个结点在一定时间内没有受到 leader 发来的心跳，则该结点认为 leader 结点已经崩溃了，故该结点将自身设为 candidate，并发起新一轮的投票。如果分布式系统中大部分结点都投该结点，则该结点赢得选举并称为 leader；如果没能获得大部分的结点的投票，则重新开始新的

一轮投票。

2.2.2 RequestVote RPC 实现

如果 RequestVoteArgs 的任期号小于当前结点的任期号，返回 false。

如果当前结点的 votedFor 为空或者与 candidateId 相同，并且候选人的日志和自己的日志一样新，则给该候选人投票。

候选人收到 reply 消息后，如果 reply 的任期号大于候选人的任期号，则将候选人置为 follower。

如果 reply 的 VoteGranted 为 true，则候选人得票数加一。判断如果得票数超过半数，则候选人成为 leader。

2.2.3 AppendEntries RPC 实现

如果 AppendEntriesAgs 的任期号小于当前结点的任期号，返回 false。

主要只用于发送心跳，并未实现日志的复制。

2.2.4 Make()的实现

首先创建 Raft 实例，并将初始状态设置为 follower，初始化其他参数。

启动一个 go routine，监听结点的状态。

如果 rf.state = follower，实现超时重新选举 leader 功能。

如果 rf.state = candidate，将任期号加一，开启一轮新的投票。启动另一个 go routine，广播投票请求。如果此时受到心跳消息，则回到 follower 状态；如果 leader 的通道传入消息，则认为候选者赢得选举，置 state 为 leader。

如果 rf.state = leader，则广播心跳消息，并等待结点答复。

3 实验结果

3.1 本地测试

执行 test_test.go 文件，测试领导者选举过程结果通过，如图 1 所示：

```
konnase@ubuntu17:~/workspace/go/src/raft$ go test -run Election
Test: initial election ...
  ... Passed
Test: election after network failure ...
  ... Passed
PASS
ok      raft    8.509s
```

图 1　测试结果

4 总结与体会

本次实验中，我了解了 raft 算法的工作原理，虽然还没有看过 Paxos 算法，也不知道 Paxos 算法究竟有多难，但是我觉得 raft 算法比较容易理解。在实现领导者选举的过程中，虽然算法描述看上去较为简单，但实现起来需要注意很多细节，比如在候选者调用 RequestVote RPC 之后，需要判断“候选者”是否任然处于候选者状态，如果不再是候选者，则不能为它加票数了。总之，实现 raft 的领导者选举这一步骤让我理解了分布式系统协同工作的原理，也深深佩服 raft 算法的创造者。

References:

[1] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm[C]// Usenix Conference on Usenix Technical Conference. USENIX Association, 2014:305-320.