Raft

In Search of an Understandable Consensus Algorithm (Extended Version)

1 简介

Raft可视化

The Secret Lives of Data

- **强Leader**：和其他一致性算法相比，Raft 使用一种更强的领导能力形式。比如，日志条目只从领导者发送给其他的服务器。这种方式简化了对复制日志的管理并且使得 Raft 算法更加易于理解。
- **领导选举**：Raft 算法使用一个随机计时器来选举领导者。这种方式只是在任何一致性算法都必须实现的心跳机制上增加了一点机制。在解决冲突的时候会更加简单快捷。
- **成员关系调整**：Raft 使用一种共同一致的方法来处理集群成员变换的问题，在这种方法下，处于调整过程中的两种不同的配置集群中大多数机器会有重叠，这就使得集群在成员变换的时候依然可以继续工作。

2 复制状态机

fzb0316@163.com

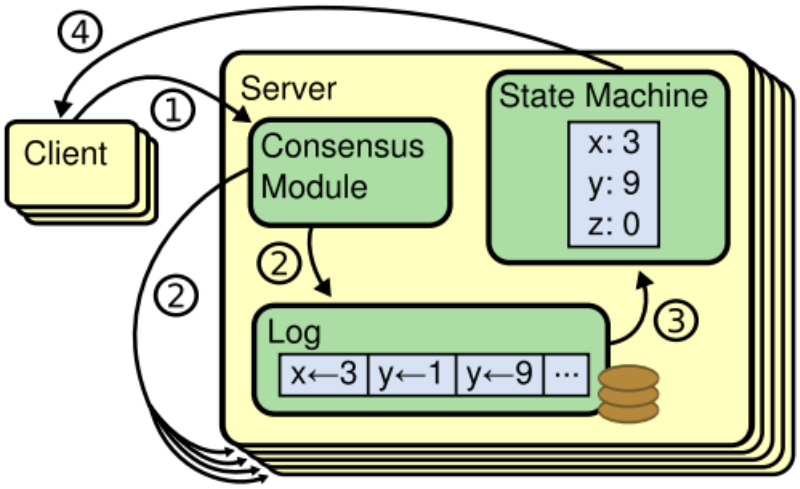


Figure 1: Replicated state machine architecture. The consensus algorithm manages a replicated log containing state machine commands from clients. The state machines process identical sequences of commands from the logs, so they produce the same outputs.

复制状态机中使用的一致性算法有以下特点：

- **安全性保证：**非拜占庭错误的情况下（网络延迟、分区、丢包、冗余、乱序等）
- **可用性：**只要大多数机器可运行并且可用相互通信、与客户端通信
- **不依赖时序来保证一致性**
- **小部分的慢节点不会影响系统整体的性能**

3 Paxos的各种不行（弱Leader）等

Paxos难以理解，而且 没有提供一个足够好的用来构建一个现实系统的基础 。

> 在Paxos算法描述和实现现实系统中间有着巨大的鸿沟。最终的系统建立在一种没有经过证明的算法之上。

4 Raft算法

算法的浓缩：

fzb0316@163.com

State

Persistent state on all servers:
(Updated on stable storage before responding to RPCs)

currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)

Volatile state on all servers:

commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied	index of highest log entry applied to state machine (initialized to 0, increases monotonically)

Volatile state on leaders:
(Reinitialized after election)

nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

fzb0316@163.com

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

fzb0316@163.com

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

fzb0316@163.com

Rules for Servers

All Servers:
- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):
- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):
- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Leaders:
- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).

4.1 算法基础

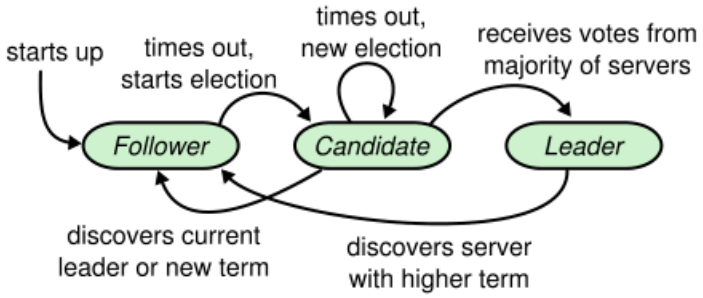


Figure 4: Server states. Followers only respond to requests from other servers. If a follower receives no communication, it becomes a candidate and initiates an election. A candidate that receives votes from a majority of the full cluster becomes the new leader. Leaders typically operate until they fail.

fzb0316@163.com

每个server有三个状态，通常情况下只有一个Leader，其余都是Follower。

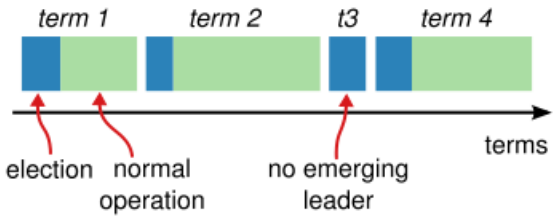


Figure 5: Time is divided into terms, and each term begins with an election. After a successful election, a single leader manages the cluster until the end of the term. Some elections fail, in which case the term ends without choosing a leader. The transitions between terms may be observed at different times on different servers.

t3时刻，选票被瓜分，这时本任期没有领导人，开始下一次的选举。

term（任期）：是连续增长的整数，是逻辑时钟

> 每一个节点存储一个当前任期号，这一编号在整个时期内单调的增长。当服务器之间通信的时候会交换当前任期号；如果一个服务器的当前任期号比其他人小，那么他会更新自己的编号到较大的编号值。如果一个候选人或者领导者发现自己的任期号过期了，那么他会立即恢复成跟随者状态。如果一个节点接收到一个包含过期的任期号的请求，那么他会直接拒绝这个请求。

通信使用RPC请求。基本的一致性算法只需要两种：请求投票（ RequestVote ）和附加条目（ AppendEntries ）　为了实现快照（Snapshot）增加了第三种RPC。

4.2 领导人选举（心跳机制）

初始状态：Follower。只要收到有效的RPC就会维持Follower状态，当选举超时时间内没有收到消息，就会触发选举行为。

选举：Follower转换为Candidate，并向其他节点发送“请求投票RPCs”，Candidate会持续保持当前状态直到（ (a) 他自己赢得了这次的选举，(b) 其他的服务器成为领导者，（c）一段时间之后没有任何一个获胜的人 ）

- **（a）赢得选举：（获得大多数投票）**成为Leader，向其他节点发送心跳包来保持地位。每个节点最多投一票。保证了同一时刻只有一个候选人可以赢得选举。
- **（b）其他服务器成为Leader**：如果这个Leader的Term不小于自己，接受，并转为Follower。
- **（c）没有获胜的**：选票被瓜分，使用随机选举超时时间法（100–300ms）来减少这种情况。每一个候选人在开始一次选举的时候会重置一个随机的选举超时时间，然后在超时时间内等待投票的结果

时间和可用性：广播时间(broadcastTime) << 选举超时时间(electionTimeout) << 平均故障时间(MTBF)

> - **广播时间** 指从一个服务器并行发送 *RPC* 给集群中其他服务器并接收响应的平均时间（0.5~20ms）
> - **选举超时时间** 即上文介绍的选举的超时时间限制（10~500ms）
> - **平均故障间隔时间** 指对于一台服务器而言，两次故障之间的平均时间（几个月甚至更长）
> - 广播时间远小于选举超时时间，是为了使 *leader* 能够发送稳定的心跳包维持管理
> - 选举超时时间远小于平均故障时间，是为了使整个系统稳定运行（ *leader* 崩溃后，系统不可用时间为选举超时时间，这个时间应在合理范围内尽量小 ）

4.3 日志复制

日志：有序序号标记的条目组成，每个条目包括创建时的任期号和需要执行的指令；被提交的日志都是持久化的，日志复制到大多数节点上的时候，日志条目就会被提交。

Leader维护当前被提交的最大的日志索引，这个索引放在附加日志RPCs中

复制流程：

> - *leader* 接收到来自 *client* 的请求
> - *leader* 将请求中的命令作为一个 *log entry* 写入本服务器的日志
> - *leader* 并行地发起 AppendEntries RPC请求给其他服务器，让它们复制这条 *log entry*
> - 当这条 *log entry* 被复制到集群中的大多数服务器（ 即成功提交 ）， *leader* 将这条 *log entry* 应用到状态机（ 即执行对应命令 ）
> - *leader* 执行命令后响应 *client*
> - *leader* 会记录最后提交的 *log entry* 的 *index* ，并在后续 AppendEntries RPC 请求（ 包含心跳包 ）中包含该 *index* ， *follower* 将此 *index* 指向的 *log entry* 应用到状态机
> - 若 *follower* 崩溃或运行缓慢或有网络丢包， *leader* 会不断重复尝试 AppendEntries RPC ， 直到所有 *follower* 都最终存储了所有 *log entry*
> - 若 *leader* 在某条 *log entry* 提交前崩溃，则 *leader* 不会对 *client* 响应，所以 *client* 会重新发送包含此命令的请求

一致性保证：（只要附加日志RPCs返回，则该节点与Leader的日志一致）

- 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。
- 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，那么他们之前的所有日志条目也全部相同。

不一致的可能和解决：

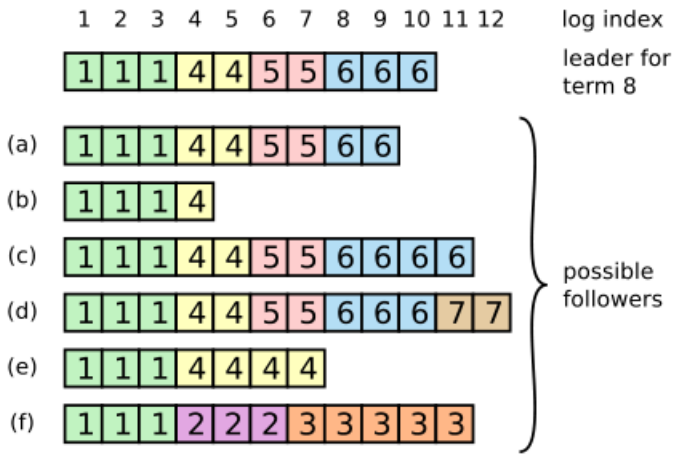


Figure 7: When the leader at the top comes to power, it is possible that any of scenarios (a–f) could occur in follower logs. Each box represents one log entry; the number in the box is its term. A follower may be missing entries (a–b), may have extra uncommitted entries (c–d), or both (e–f). For example, scenario (f) could occur if that server was the leader for term 2, added several entries to its log, then crashed before committing any of them; it restarted quickly, became leader for term 3, and added a few more entries to its log; before any of the entries in either term 2 or term 3 were committed, the server crashed again and remained down for several terms.

> - *leader* 对每一个 *follower* 都维护了一个 nextIndex ，表示下一个需要发送给 *follower* 的 *log entry* 的 *index*
> - *leader* 刚上任后，会将所有 *follower* 的 *nextIndex* 初始化为自己的最后一条 *log entry* 的 *index* 加一（ 上图中的 *11* ）
> - 如果一个 AppendEntries RPC 被 *follower* 拒绝后（ *leader* 和 *follower* 不一致 ），*leader* 减小 *nextIndex* 值重试（ *prevLogIndex* 和 *prevLogTerm* 也会对应改变 ）
> - 最终 *nextIndex* 会在某个位置使得 *leader* 和 *follower* 达成一致，此时，AppendEntries RPC 成功，将 *follower* 中的冲突 *log entry* 删除并加上 *leader* 的 *log entiry*

上述引用的小优化：当 AppendEntries RPC 被拒绝时返回冲突 *log entry* 的 *term* 和 属于该 *term* 的 *log entry* 的最早 *index* 。*leader* 重新发送请求时减小 *nextIndex* 越过那个 *term* 冲突的所有 *log entry* 。这样就变成了每个 *term* 需要一次 *RPC* 请求，而非每个 *log entry* 一次 。

4.4 算法完善（保证任何Leader完整特性）

选举限制：Leader必须存储所有已经提交的日志条目。相比于其他算法，Raft在投票时加以限制， 通过比较两份日志中最后一条日志条目的索引值和任期号定义谁的日志比较新 ，Follower只会给比较新的Candidate投票，否则不投票而且不重置自己的选举超时计时器。

4.5 Follower和Candidate崩溃

出现这种情况，他们的RPC就会失败，Raft处理这种失败的方法就是无限地重试。

5 日志压缩（SnapShot）

日志的不断增长会引起空间占用太大、服务器重启时重放日志会花费太多的时间

快照的意义如下图所示：

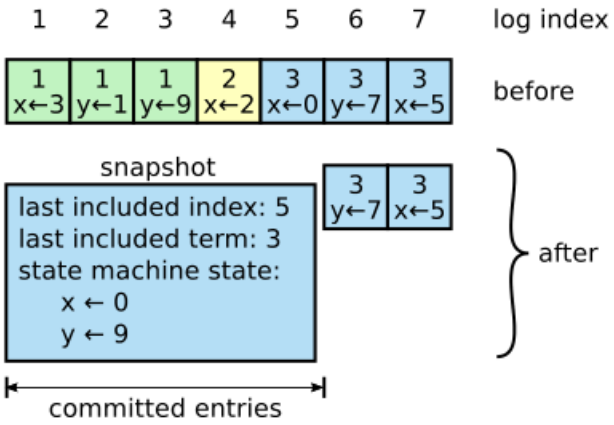
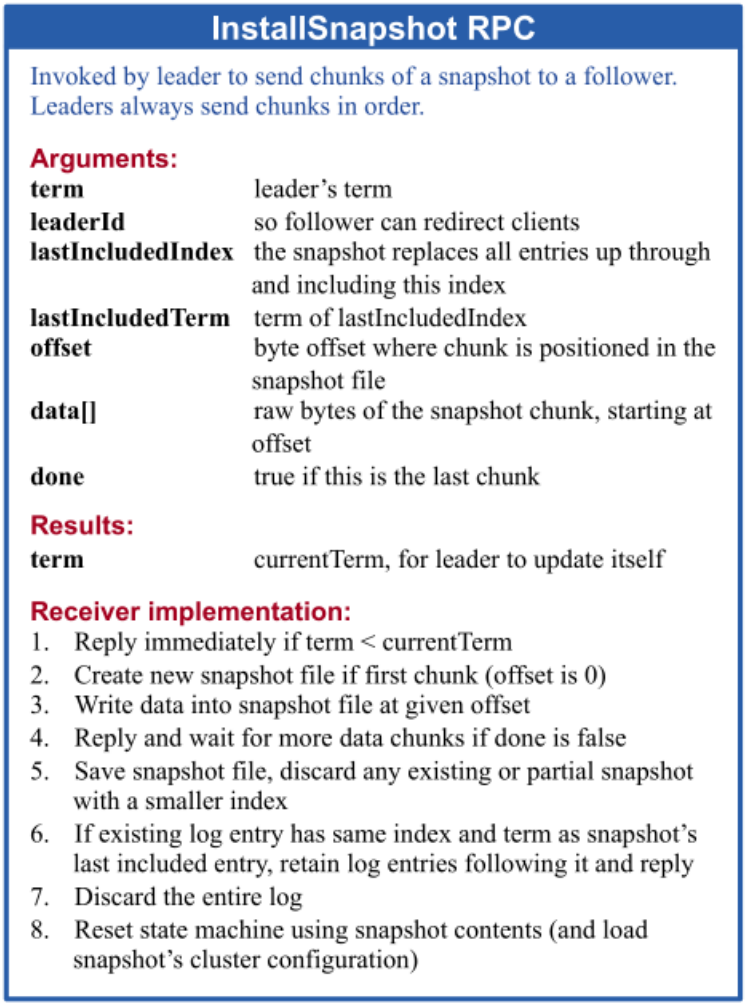


Figure 12: A server replaces the committed entries in its log (indexes 1 through 5) with a new snapshot, which stores just the current state (variables *x* and *y* in this example). The snapshot's last included index and term serve to position the snapshot in the log preceding entry 6.

- 状态机状态（以数据库举例，则快照记录的是数据库中的表）
- 最后被包含的 *index*： 被快照取代的最后一个 *log entry* 的 *index*
- 最后被包含的 *term*：被快照取代的最后一个 *log entry* 的 *term*

通常由每个服务器独立地创建快照，但 *leader* 会偶尔发送快照给一些落后的 *follower* 。这通常发生在当 *leader* 已经丢弃了下一条需要发送给 *follower* 的 *log entry* 时



InstallSnapshot RPC

Invoked by leader to send chunks of a snapshot to a follower. Leaders always send chunks in order.

Arguments:

term	leader's term
leaderId	so follower can redirect clients
lastIncludedIndex	the snapshot replaces all entries up through and including this index
lastIncludedTerm	term of lastIncludedIndex
offset	byte offset where chunk is positioned in the snapshot file
data[]	raw bytes of the snapshot chunk, starting at offset
done	true if this is the last chunk

Results:

term	currentTerm, for leader to update itself

Receiver implementation:

1. Reply immediately if term < currentTerm
2. Create new snapshot file if first chunk (offset is 0)
3. Write data into snapshot file at given offset
4. Reply and wait for more data chunks if done is false
5. Save snapshot file, discard any existing or partial snapshot with a smaller index
6. If existing log entry has same index and term as snapshot's last included entry, retain log entries following it and reply
7. Discard the entire log
8. Reset state machine using snapshot contents (and load snapshot's cluster configuration)

Figure 13: A summary of the InstallSnapshot RPC. Snapshots are split into chunks for transmission; this gives the follower a sign of life with each chunk, so it can reset its election timer.

6 客户端交互

- *client* 启动时，会随机挑选一个服务器进行通信
- 若该服务器是 *follower*，则拒绝请求并提供它最近接收到的 *leader* 的信息给 *client*
- *client* 发送请求给 *leader*
- 若 *leader* 已崩溃，*client* 请求会超时，之后再随机挑选服务器进行通信
- 若raft收到一条命令多次（使用RequestId和ClientId判断），不会重复执行
- 只读操作可以直接处理而无需记录日志