Understanding Raft

Stephan Merz

University of Lorraine, CNRS, Inria, LORIA, Nancy, France

IFIP Working Group 2.3
Southampton, September 7, 2026

Motivation

- Distributed consensus in the presence of faults
  - fundamental problem: servers agree on (a sequence of) proposals
  - famously unsolvable for asynchronous communication and crash faults (FLP 1985)
  - existing algorithms ensure safety, rely on sufficient availability for liveness

Motivation

- Distributed consensus in the presence of faults
  - fundamental problem: servers agree on (a sequence of) proposals
  - famously unsolvable for asynchronous communication and crash faults (FLP 1985)
  - existing algorithms ensure safety, rely on sufficient availability for liveness
- Raft algorithm
  - introduced as an "easier to understand" alternative to Paxos
  - fundamentally designed as an iterated consensus algorithm
  - every server maintains a sequence ("log") of entries
  - logs converge to common prefixes over time

D. Ongaro, J.K. Ousterhout: In search of an understandable consensus algorithm. Usenix ATC 2014.

Formal Verification of Raft

- TLA$^+$ specification written by Ongaro
  - precise blueprint for implementing the protocol
  - level of abstraction close to the implementation
  - validation by simulation and partial proof of some properties
  - Raft paper and Ongaro's PhD thesis present a behavioral correctness proof

Formal Verification of Raft

- TLA$^+$ specification written by Ongaro
  - precise blueprint for implementing the protocol
  - level of abstraction close to the implementation
  - validation by simulation and partial proof of some properties
  - Raft paper and Ongaro's PhD thesis present a behavioral correctness proof
- Mechanized correctness proofs based on inductive invariants
  - formal proof in Verdi framework: $\sim$ 50k lines of Rocq proofs
  - recent proof based on Ongaro's specification: $\sim$ 30k lines of TLA$^+$ proofs
  - heroic achievements, but difficult to understand and extend

D. Woos et al.: Planning for change in a formal verification of the Raft consensus protocol. CPP 2015.
O. Marcu: announcement on LinkedIn, 2026.

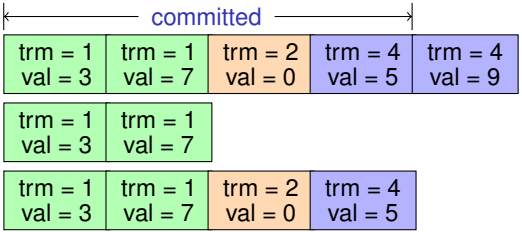
Formal Verification of Raft

- TLA$^+$ specification written by Ongaro
  - precise blueprint for implementing the protocol
  - level of abstraction close to the implementation
  - validation by simulation and partial proof of some properties
  - Raft paper and Ongaro's PhD thesis present a behavioral correctness proof
- Mechanized correctness proofs based on inductive invariants
  - formal proof in Verdi framework: $\sim$ 50k lines of Rocq proofs
  - recent proof based on Ongaro's specification: $\sim$ 30k lines of TLA$^+$ proofs
  - heroic achievements, but difficult to understand and extend
- Objective: model and verify Raft at a higher level of abstraction

D. Woos et al.: Planning for change in a formal verification of the Raft consensus protocol. CPP 2015.
O. Marcu: announcement on LinkedIn, 2026.

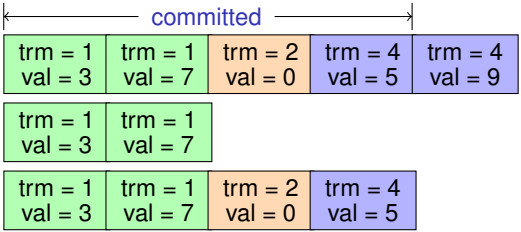
Overview of Raft

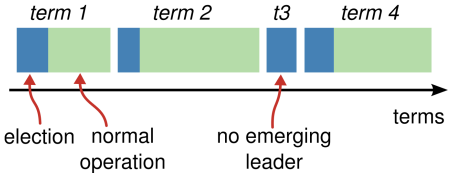
- Servers and their logs
  - every server maintains a sequence (“log”) of entries containing proposed values
  - a prefix of the log is committed (stable), the rest may be subject to modification



Overview of Raft

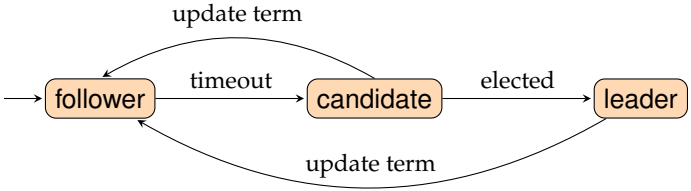
- Servers and their logs
  - every server maintains a sequence (“log”) of entries containing proposed values
  - a prefix of the log is committed (stable), the rest may be subject to modification
- Leaders and followers
  - Raft operates over successive terms managed by a leader
  - leader is responsible for introducing new entries and for committing entries
  - if a leader is suspected to have failed, a new leader is elected





Outline

Lifecycle of Server States



- Elections and terms
  - leaders are elected for a term
  - follower may suspect the leader to have failed and become a candidate for a higher term
  - servers only vote for candidates at strictly higher terms
  - a candidate is elected when it receives votes from a quorum of servers
  - whenever a server learns of the existence of a higher term, it becomes a follower

Timeout and Voting in TLA$^+$

- A follower or candidate starts a new election and votes for itself

$$
\begin{aligned}
Timeout(s) \triangleq\ & \land role[s] \in \{\text{“follower”}, \text{“candidate”}\} \\
& \land role' = [role \ \text{EXCEPT}\ ![s] = \text{“candidate”}] \\
& \land \exists t \in Term : \land t > term[s] \land term' = [term \ \text{EXCEPT}\ ![s] = t] \\
& \qquad\qquad\ \land ballots' = [ballots \ \text{EXCEPT}\ ![s] = @ \cup \{\langle s, t\rangle\}]
\end{aligned}
$$

Timeout and Voting in TLA$^+$

- A follower or candidate starts a new election and votes for itself

$$
\begin{array}{l}
Timeout(s) \triangleq \land\ role[s] \in \{\text{“follower”}, \text{“candidate”}\} \\
\qquad\qquad\ \land\ role' = [role\ \text{EXCEPT}\ ![s] = \text{“candidate”}] \\
\qquad\qquad\ \land\ \exists t \in Term : \land\ t > term[s] \land term' = [term\ \text{EXCEPT}\ ![s] = t] \\
\qquad\qquad\qquad\qquad\ \land\ ballots' = [ballots\ \text{EXCEPT}\ ![s] = @ \cup \{\langle s, t\rangle\}]
\end{array}
$$

- A server votes for a candidate at a higher term

$$
\begin{array}{l}
Vote(s) \triangleq \exists cdt \in Server \setminus \{s\} : \text{LET}\ ct \triangleq term[cdt]\ \text{IN} \\
\quad \land\ term[s] < ct \\
\quad \land\ term' = [term\ EXCEPT\ ![s] = ct] \\
\quad \land\ role' = [role\ \text{EXCEPT}\ ![s] = \text{“follower”}] \\
\quad \land\ ballots' = [ballots\ \text{EXCEPT}\ ![s] = @ \cup \{\langle cdt, ct\rangle\}]
\end{array}
$$

Electing a Leader and Term Update

- A candidate is elected leader when it obtains a quorum of votes

$$
\begin{aligned}
ElectLeader(s) \triangleq\ & \land\ role[s] = \text{“candidate”} \\
& \land\ \exists Q \in Quorum : \forall vt \in Q : \langle s, term[s] \rangle \in ballots[vt] \\
& \land\ role' = [role \ \text{EXCEPT}\ ![s] = \text{“leader”}] \\
& \land\ term' = term
\end{aligned}
$$

Electing a Leader and Term Update

- A candidate is elected leader when it obtains a quorum of votes

$$
\begin{aligned}
ElectLeader(s) \triangleq\ & \land\ role[s] = \text{“candidate”} \\
& \land\ \exists Q \in Quorum : \forall vt \in Q : \langle s, term[s] \rangle \in ballots[vt] \\
& \land\ role' = [role \ \text{EXCEPT}\ ![s] = \text{“leader”}] \\
& \land\ term' = term
\end{aligned}
$$

- A server becomes follower when it learns of a server at a higher term

$$
\begin{aligned}
&UpdateTerm(s) \triangleq\ \exists srv \in Server : \\
&\quad \land\ term[srv] > term[s] \\
&\quad \land\ term' = [term \ \text{EXCEPT}\ ![s] = term[srv]] \\
&\quad \land\ role' = [role \ \text{EXCEPT}\ ![s] = \text{“follower”}]
\end{aligned}
$$

Election Invariant

$$BackedByQuorum(srv, t) \triangleq \exists Q \in Quorum : \forall vt \in Q : \langle srv, t \rangle \in ballots[vt]$$

$$\begin{aligned} ElectionInv \triangleq\ & \land \forall cdt, srv \in Server : \forall t \in Term : \langle cdt, t \rangle \in ballots[srv] \Rightarrow \\ & \qquad \land 0 < t \land t \leq term[cdt] \land t \leq term[srv] \\ & \qquad \land \forall b \in ballots[srv] : b[2] = t \Rightarrow b = \langle cdt, t \rangle \\ & \land \forall srv \in Server : role[srv] \in \{\text{“candidate”}, \text{“follower”}\} \Rightarrow term[srv] > 0 \\ & \land \forall ldr \in Server : role[ldr] = \text{“leader”} \Rightarrow BackedByQuorum(ldr, term[ldr]) \end{aligned}$$

- a server votes only for one candidate at any given term
- the term of any ballot is a lower bound for the terms of the voter and the candidate
- leaders are backed by a quorum of voters

Election Invariant

$$BackedByQuorum(srv, t) \triangleq \exists Q \in Quorum : \forall vt \in Q : \langle srv, t \rangle \in ballots[vt]$$

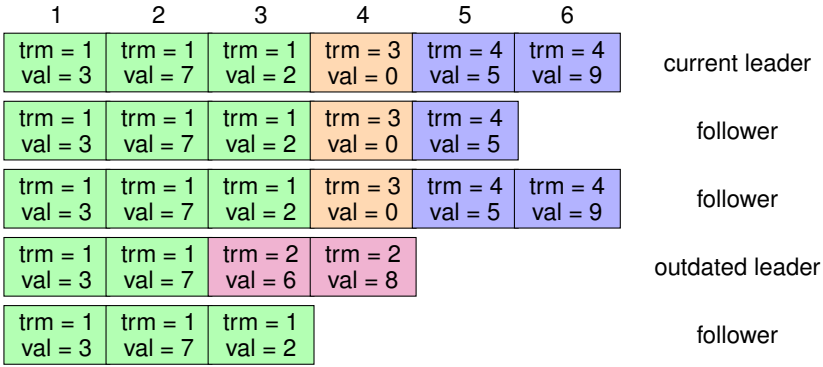
$$\begin{aligned} ElectionInv \triangleq\ & \wedge \forall cdt, srv \in Server : \forall t \in Term : \langle cdt, t \rangle \in ballots[srv] \Rightarrow \\ & \qquad \wedge 0 < t \wedge t \leq term[cdt] \wedge t \leq term[srv] \\ & \qquad \wedge \forall b \in ballots[srv] : b[2] = t \Rightarrow b = \langle cdt, t \rangle \\ & \wedge \forall srv \in Server : role[srv] \in \{\text{“candidate”}, \text{“follower”}\} \Rightarrow term[srv] > 0 \\ & \wedge \forall ldr \in Server : role[ldr] = \text{“leader”} \Rightarrow BackedByQuorum(ldr, term[ldr]) \end{aligned}$$

- a server votes only for one candidate at any given term
- the term of any ballot is a lower bound for the terms of the voter and the candidate
- leaders are backed by a quorum of voters
- since any two quorums intersect, there can only be one leader per term

$$\begin{aligned} \text{LEMMA } BBQSameTerm \triangleq\ & \text{ASSUME } ElectionInv, \text{ NEW } s1, s2 \in Server, \text{ NEW } t \in Term, \\ & \qquad BackedByQuorum(s1, t), BackedByQuorum(s2, t) \\ & \text{PROVE } s1 = s2 \end{aligned}$$

Outline

Servers and their Logs



- entries are tagged with the term during which they were added
- leaders append entries to their logs
- followers copy entries from their leader

Adding an Entry to the Log of a Leader

A leader serves a client request

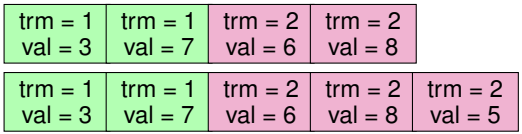
$$\begin{aligned} AppendEntry(s) \triangleq\ & \wedge role[s] = \text{“leader”} \\ & \wedge \exists v \in Value : entries' = [entries \ \text{EXCEPT} \ ![s] = \\ & \qquad Append(@, [val \mapsto v, trm \mapsto term[s]])] \end{aligned}$$

Adding an Entry to the Log of a Leader

A leader serves a client request

$$AppendEntry(s) \triangleq \land role[s] = \text{“leader”}$$
$$\land \exists v \in Value : entries' = [entries \ \text{EXCEPT} \ ![s] =$$
$$Append(@, [val \mapsto v, trm \mapsto term[s]])]$$

- the entry is tagged with the leader's term and appended to the log



Updating the Log of a Follower

Find the first entry at which the logs differ and copy the leader's entry

$$
\begin{aligned}
&LearnEntry(s) \triangleq \exists ldr \in Server : \\
&\quad \land role[s] = \text{“follower”} \\
&\quad \land BackedByQuorum(ldr, term[s]) \\
&\quad \land term[ldr] = term[s] \\
&\quad \land \exists n \in 1 \,..\, Min(Len(entries[s]) + 1, Len(entries[ldr])) : \\
&\qquad \land n - 1 \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n-1].trm = entries[ldr][n-1].trm \\
&\qquad \land n \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[ldr][n].trm \\
&\qquad \land entries' = [entries \ \text{EXCEPT} \ ![s] = Append(SubSeq(entries[s], 1, n-1), entries[ldr][n])]
\end{aligned}
$$

Updating the Log of a Follower

Find the first entry at which the logs differ and copy the leader's entry

$$
\begin{aligned}
&LearnEntry(s) \triangleq \exists ldr \in Server : \\
&\quad \wedge\ role[s] = \text{“follower”} \\
&\quad \wedge\ BackedByQuorum(ldr, term[s]) \\
&\quad \wedge\ term[ldr] = term[s] \\
&\quad \wedge\ \exists n \in 1\,..\,Min(Len(entries[s]) + 1, Len(entries[ldr])) : \\
&\qquad \wedge\ n - 1 \in 1\,..\,Len(entries[s]) \Rightarrow entries[s][n-1].trm = entries[ldr][n-1].trm \\
&\qquad \wedge\ n \in 1\,..\,Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[ldr][n].trm \\
&\qquad \wedge\ entries' = [entries \ \text{EXCEPT}\ ![s] = Append(SubSeq(entries[s], 1, n-1), entries[ldr][n])]
\end{aligned}
$$

trm = 1 val = 3	trm = 1 val = 7	trm = 1 val = 2	trm = 3 val = 0	trm = 4 val = 5	trm = 4 val = 9

leader

trm = 1 val = 3	trm = 1 val = 7	trm = 2 val = 6	trm = 2 val = 8	trm = 2 val = 5

follower, before

trm = 1 val = 3	trm = 1 val = 7	trm = 1 val = 2

follower, after

LearnEntry may overwrite logs

Log Invariants (1)

- Every entry was introduced by a leader for its term

$$
\begin{array}{l}
\mathit{EntryHasCreator} \triangleq \forall s \in \mathit{Server} : \forall i \in 1 \mathinner{..} \mathit{Len}(\mathit{entries}[s]) : \exists c \in \mathit{Server} : \\
\quad \land \mathit{BackedByQuorum}(c, \mathit{entries}[s][i].\mathit{trm}) \\
\quad \land \lor \mathit{role}[c] = \text{“leader”} \land \mathit{term}[c] = \mathit{entries}[s][i].\mathit{trm} \\
\qquad \lor \mathit{term}[c] > \mathit{entries}[s][i].\mathit{trm}
\end{array}
$$

- A leader for term t holds all entries of term t that appear anywhere

$$
\begin{array}{l}
\mathit{InLog}(s, i, e) \triangleq i \in 1 \mathinner{..} \mathit{Len}(\mathit{entries}[s]) \land \mathit{entries}[s][i] = e \\
\mathit{LeaderHasEntries} \triangleq \forall \mathit{ldr}, s \in \mathit{Server} : \forall i \in 1 \mathinner{..} \mathit{Len}(\mathit{entries}[s]) : \\
\quad \mathit{role}[\mathit{ldr}] = \text{“leader”} \land \mathit{entries}[s][i].\mathit{trm} = \mathit{term}[\mathit{ldr}] \Rightarrow \mathit{InLog}(\mathit{ldr}, i, \mathit{entries}[s][i])
\end{array}
$$

Log Invariants (2)

- Entry terms are bounded by the term of the server and monotonic

$$EntryTerm \triangleq \forall s \in Server : \forall i \in 1..Len(entries[s]) : entries[s][i].trm \leq term[s]$$

$$EntryTermMonotonic \triangleq \forall s \in Server : \forall i, j \in 1..Len(entries[s]) : i \leq j \Rightarrow entries[s][i].trm \leq entries[s][j].trm$$

- The position and term of an entry uniquely determines the prefix of the log

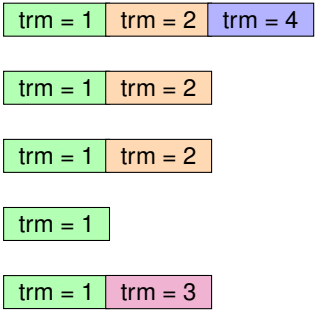
$$LogMatching \triangleq \forall s1, s2 \in Server : \forall i \in 1..Min(Len(entries[s1], Len(entries[s2]))) : entries[s1][i].trm = entries[s2][i].trm \Rightarrow \forall j \in 1..i : entries[s1][j] = entries[s2][j]$$

- These invariants are inductive relative to the election invariant

Outline

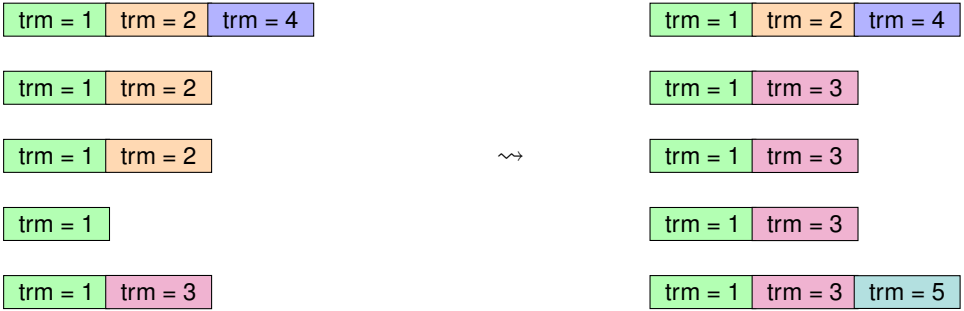
When does an Entry Become Stable?

- Eventually, entries should stabilize across the network
  - idea: commit an entry when it has been replicated on a quorum of servers
  - Can the entry of term 2 be declared committed?



When does an Entry Become Stable?

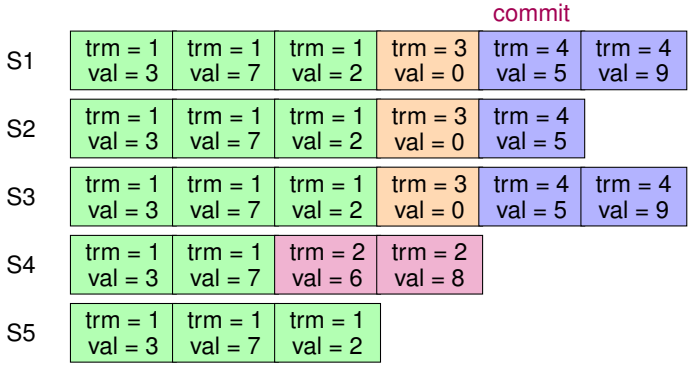
- Eventually, entries should stabilize across the network
  - idea: commit an entry when it has been replicated on a quorum of servers
  - Can the entry of term 2 be declared committed? No!



- Committing entries is delicate

Primary Commits

- A leader may commit an entry introduced in its current term . . .
  - . . . provided that entry is backed by a quorum of followers at that term
  - the entire prefix of the log is implicitly committed as well



- Must prevent servers S4 and S5 from being elected leaders!

The Voting Proviso

- Add a precondition to the *Vote* action

$$
\begin{array}{l}
lastEntryTerm(es) \triangleq \text{IF } es = \langle\rangle \text{ THEN } 0 \text{ ELSE } es[Len(es)].trm \\
Vote(s) \triangleq \exists cdt \in Server \setminus \{s\} : \text{LET } ct \triangleq term[cdt] \text{ IN} \\
\quad \land\ term[s] < ct \land \forall b \in ballots[s] : b[2] < ct \\
\quad \land\ \lor\ lastEntryTerm(entries[s]) < lastEntryTerm(entries[cdt]) \\
\qquad \lor\ \land\ lastEntryTerm(entries[s]) = lastEntryTerm(entries[cdt]) \\
\qquad\quad \land\ Len(entries[s]) \leq Len(entries[cdt]) \\
\quad \land\ term' = [term \ EXCEPT\ ![s] = ct] \\
\quad \land\ role' = [role \ \text{EXCEPT } ![s] = \text{“follower”}] \\
\quad \land\ ballots' = [ballots \ \text{EXCEPT } ![s] = @ \cup \{\langle cdt, ct\rangle\}]
\end{array}
$$

- Must explain why this is enough, given that logs are not stable!

Leader Commit

- A leader commits an entry from the current term backed by a quorum

$$
\begin{array}{l}
LeaderCommit(s) \triangleq \\
\quad \land\ role[s] = \text{“leader”} \\
\quad \land\ \exists n \in (commitIndex[s] + 1) \,..\, Len(entries[s]) : \\
\qquad \land\ entries[s][n].trm = term[s] \\
\qquad \land\ \exists Q \in Quorum : \land\ \forall sq \in Q : InLog(sq, n, entries[s][n]) \land term[sq] = term[s] \\
\qquad\qquad\qquad\qquad\ \land\ commitQuorum' = [commitQuorum \ \text{EXCEPT}\ ![s][n] = Q] \\
\qquad \land\ commitIndex' = [commitIndex \ \text{EXCEPT}\ ![s] = n]
\end{array}
$$

  - *commitQuorum* is a history variable and records the backing quorum for the proof
  - *commitIndex* indicates for each server the committed prefix of the log

- Adaptation of action *LearnEntry*
  - if necessary, adjust *commitIndex* so that it doesn't exceed the length of the log

Push Information on Committed Entries to Followers

- Followers learn about committed entries from other servers

$$
\begin{aligned}
&GossipCommit(s) \triangleq \exists srv \in Server : \exists n \in (commitIndex[s]+1)\,..\,commitIndex[srv] : \\
&\quad \wedge\ role[s] = \text{“follower”} \\
&\quad \wedge\ term[s] \leq term[srv] \\
&\quad \wedge\ InLog(s, n, entries[srv][n]) \\
&\quad \wedge\ term' = [term \ \text{EXCEPT}\ ![s] = term[srv]] \\
&\quad \wedge\ commitIndex' = [commitIndex \ \text{EXCEPT}\ ![s] = n]
\end{aligned}
$$

  - any entry that is committed somewhere and present in the log can be committed

Invariants on Committed Entries (1)

- $commitQuorum[s][i]$ is either empty or contains a quorum, and s is a leader

$$
\begin{aligned}
CQType \triangleq\ & \forall s \in Server, i \in Index : commitQuorum[s][i] \neq \emptyset \Rightarrow \\
& \land\ i \in 1\,..\,commitIndex[s] \\
& \land\ commitQuorum[s] \in Quorum \\
& \land\ BackedByQuorum(s, entries[s][i].trm)
\end{aligned}
$$

- Every committed entry is dominated by a leader commit

$$
\begin{aligned}
& PrimaryCommit(ldr, j) \triangleq commitQuorum[ldr][j] \in Quorum \\
& CommitHasPrimary \triangleq \\
& \forall s \in Server : \forall i \in 1\,..\,commitIndex[s] : \exists ldr \in Server : \exists j \in i\,..\,commitIndex[ldr] : \\
& \quad \land\ PrimaryCommit(ldr, j) \\
& \quad \land\ InLog(ldr, i, entries[s][i]) \\
& \quad \land\ term[s] \geq entries[ldr][j].trm
\end{aligned}
$$

Invariants on Committed Entries (2)

- All members of a commit quorum only vote for servers that contain the entry

$$
\begin{array}{l}
CommitQuorum \triangleq \forall ldr \in Server : \forall i \in 1 \,..\, Len(entries[ldr]) : PrimaryCommit(ldr, i) \Rightarrow \\
\quad \forall s \in commitQuorum[ldr][i] : \wedge\, InLog(s, i, entries[ldr][i]) \wedge term[s] \geq entries[ldr][i].trm \\
\qquad\qquad\qquad\qquad \wedge \forall cdt \in Server,\ t \in Term : \\
\qquad\qquad\qquad\qquad\qquad t > entries[ldr][i].trm \wedge \langle cdt, t \rangle \in ballots[s] \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad InLog(cdt, i, entries[ldr][i])
\end{array}
$$

- Servers that are more up to date than a leader commit contain the committed entry

$$
\begin{array}{l}
MoreUpToDateHasCommit \triangleq \forall ldr, s \in Server : \forall i \in 1 \,..\, Len(entries[ldr]) : \forall j \in 1 \,..\, Len(entries[s]) : \\
\quad PrimaryCommit(ldr, i) \wedge \vee\, entries[s][j].trm > entries[ldr][i].trm \\
\qquad\qquad\qquad\qquad \vee\, entries[s][j].trm = entries[ldr][i].trm \wedge j > i \\
\quad \Rightarrow\ InLog(s, i, entries[ldr][i])
\end{array}
$$

- Invariants focus on primary commits

Leader Completeness

- Consequence of the commit invariant
  - If a server is a leader for a term higher than that of a primary commit, it must contain the committed entry.

$$
\begin{aligned}
\mathit{LeaderComplete} \triangleq\ & \forall \mathit{ld}1, \mathit{ld}2 \in \mathit{Server} : \forall i \in 1 \mathinner{..} \mathit{Len}(\mathit{entries}[\mathit{ld}1]) : \forall t \in \mathit{Term} : \\
& \quad \land\ \mathit{PrimaryCommit}(\mathit{ld}1, i) \\
& \quad \land\ \mathit{BackedByQuorum}(\mathit{ld}2, t) \\
& \quad \land\ t > \mathit{entries}[\mathit{ld}1][i].\mathit{trm} \\
& \Rightarrow\ \mathit{InLog}(\mathit{ld}2, i, \mathit{entries}[\mathit{ld}1][i])
\end{aligned}
$$

- Therefore, a primary commit can never be overwritten by *LearnEntry*
  - the Raft paper can be read as asserting that property for all committed entries
  - such an entry may in fact be overwritten and restored later

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in \mathit{Server}$ and $i \in 1\,..\,\mathit{commitIndex}[s]$, define $ci \triangleq \mathit{commitIndex}[s]$.

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in Server$ and $i \in 1\,..\,commitIndex[s]$, define $ci \triangleq commitIndex[s]$.
By *CommitHasPrimary*, pick $ldr \in Server$ and $j \in ci\,..\,commitIndex[ldr]$ such that:
$PrimaryCommit(ldr, j)$, $InLog(ldr, ci, entries[s][ci])$, $term[s] \geq entries[ldr][j].trm$.

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in Server$ and $i \in 1 \,..\, commitIndex[s]$, define $ci \triangleq commitIndex[s]$.

By *CommitHasPrimary*, pick $ldr \in Server$ and $j \in ci \,..\, commitIndex[ldr]$ such that:
$PrimaryCommit(ldr, j)$, $InLog(ldr, ci, entries[s][ci])$, $term[s] \geq entries[ldr][j].trm$.

The only action potentially overwriting $entries[s][i]$ is *LearnEntry*(s), so consider
$srv \in Server$, $BackedByQuorum(srv, term[s])$, $term[srv] = term[s]$,
$n \in Min(Len(entries[s]) + 1, Len(entries[srv]))$ s.t.
$n \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[srv][n].trm$.

Assume moreover that $n \leq ci$ (otherwise $entries[s][i]$ is preserved).

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in Server$ and $i \in 1 \,..\, commitIndex[s]$, define $ci \triangleq commitIndex[s]$.

By *CommitHasPrimary*, pick $ldr \in Server$ and $j \in ci \,..\, commitIndex[ldr]$ such that:
$PrimaryCommit(ldr, j)$, $InLog(ldr, ci, entries[s][ci])$, $term[s] \geq entries[ldr][j].trm$.

The only action potentially overwriting $entries[s][i]$ is $LearnEntry(s)$, so consider
$srv \in Server$, $BackedByQuorum(srv, term[s])$, $term[srv] = term[s]$,
$n \in Min(Len(entries[s]) + 1, Len(entries[srv]))$ s.t.
$n \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[srv][n].trm$.

Assume moreover that $n \leq ci$ (otherwise $entries[s][i]$ is preserved).

It follows that $term[srv] \geq entries[ldr][j].trm$. We prove $InLog(srv, j, entries[ldr][j])$.

- If $term[srv] > entries[ldr][j].trm$, this follows from *LeaderComplete*.
- If $term[srv] = entries[ldr][j].trm$, then $srv = ldr$ by *BBQSameTerm* and *CQType*.

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in Server$ and $i \in 1 \,..\, commitIndex[s]$, define $ci \triangleq commitIndex[s]$.

By *CommitHasPrimary*, pick $ldr \in Server$ and $j \in ci \,..\, commitIndex[ldr]$ such that:
$PrimaryCommit(ldr, j)$, $InLog(ldr, ci, entries[s][ci])$, $term[s] \geq entries[ldr][j].trm$.

The only action potentially overwriting $entries[s][i]$ is $LearnEntry(s)$, so consider
$srv \in Server$, $BackedByQuorum(srv, term[s])$, $term[srv] = term[s]$,
$n \in Min(Len(entries[s]) + 1, Len(entries[srv]))$ s.t.
$n \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[srv][n].trm$.

Assume moreover that $n \leq ci$ (otherwise $entries[s][i]$ is preserved).

It follows that $term[srv] \geq entries[ldr][j].trm$. We prove $InLog(srv, j, entries[ldr][j])$.

- If $term[srv] > entries[ldr][j].trm$, this follows from *LeaderComplete*.
- If $term[srv] = entries[ldr][j].trm$, then $srv = ldr$ by *BBQSameTerm* and *CQType*.

By *LogMatching* and $ci \leq j$, it follows that $InLog(srv, ci, entries[s][ci])$.

No Entry in the Committed Prefix is Overwritten

- Entries that a server knows to be committed are stable

Proof. Assume $s \in Server$ and $i \in 1 \,..\, commitIndex[s]$, define $ci \triangleq commitIndex[s]$.

By *CommitHasPrimary*, pick $ldr \in Server$ and $j \in ci \,..\, commitIndex[ldr]$ such that:
$PrimaryCommit(ldr, j)$, $InLog(ldr, ci, entries[s][ci])$, $term[s] \geq entries[ldr][j].trm$.

The only action potentially overwriting $entries[s][i]$ is $LearnEntry(s)$, so consider
$srv \in Server$, $BackedByQuorum(srv, term[s])$, $term[srv] = term[s]$,
$n \in Min(Len(entries[s]) + 1, Len(entries[srv]))$ s.t.
$n \in 1 \,..\, Len(entries[s]) \Rightarrow entries[s][n].trm \neq entries[srv][n].trm$.

Assume moreover that $n \leq ci$ (otherwise $entries[s][i]$ is preserved).

It follows that $term[srv] \geq entries[ldr][j].trm$. We prove $InLog(srv, j, entries[ldr][j])$.

- If $term[srv] > entries[ldr][j].trm$, this follows from *LeaderComplete*.
- If $term[srv] = entries[ldr][j].trm$, then $srv = ldr$ by *BBQSameTerm* and *CQType*.

By *LogMatching* and $ci \leq j$, it follows that $InLog(srv, ci, entries[s][ci])$.

Using again *LogMatching* and $n \leq ci$, we have $InLog(srv, n, entries[s][n])$, contradiction. QED

Main Safety Property

- Any two servers agree on committed entries at the same position.

$$
\begin{array}{lll}
\text{THEOREM } \mathit{Safety} & \triangleq & \text{ASSUME } \mathit{TypeOK},\ \mathit{LogInv},\ \mathit{CommitInv}, \\
& & \quad \text{NEW } s1 \in \mathit{Server},\ \text{NEW } s2 \in \mathit{Server}, \\
& & \quad \text{NEW } i \in 1\,..\,\mathit{Min}(\mathit{commitIndex}[s1], \mathit{commitIndex}[s2]) \\
& & \text{PROVE } \quad \mathit{entries}[s1][i] = \mathit{entries}[s2][i]
\end{array}
$$

Main Safety Property

- Any two servers agree on committed entries at the same position.

$$\begin{array}{lll} \text{THEOREM } \mathit{Safety} & \triangleq & \text{ASSUME } \mathit{TypeOK},\ \mathit{LogInv},\ \mathit{CommitInv}, \\ & & \quad \text{NEW } s1 \in \mathit{Server},\ \text{NEW } s2 \in \mathit{Server}, \\ & & \quad \text{NEW } i \in 1\,..\,\mathit{Min}(\mathit{commitIndex}[s1], \mathit{commitIndex}[s2]) \\ & & \text{PROVE } \ \mathit{entries}[s1][i] = \mathit{entries}[s2][i] \end{array}$$

Proof. Using *CommitHasPrimary* and *LogMatching*, pick
$ld1 \in \mathit{Server}$ and $Q1 \in \mathit{Quorum}$ such that $\mathit{InLog}(srv, i, \mathit{entries}[s1][i]$ for all $srv \in Q1$ and
$ld2 \in \mathit{Server}$ and $Q2 \in \mathit{Quorum}$ such that $\mathit{InLog}(srv, i, \mathit{entries}[s2][i]$ for all $srv \in Q2$.

Main Safety Property

- Any two servers agree on committed entries at the same position.

THEOREM $Safety \triangleq$ ASSUME $TypeOK$, $LogInv$, $CommitInv$,
NEW $s1 \in Server$, NEW $s2 \in Server$,
NEW $i \in 1 \,..\, Min(commitIndex[s1], commitIndex[s2])$
PROVE $entries[s1][i] = entries[s2][i]$

Proof. Using *CommitHasPrimary* and *LogMatching*, pick
$ld1 \in Server$ and $Q1 \in Quorum$ such that $InLog(srv, i, entries[s1][i]$ for all $srv \in Q1$ and
$ld2 \in Server$ and $Q2 \in Quorum$ such that $InLog(srv, i, entries[s2][i]$ for all $srv \in Q2$.
Pick some $srv \in Q1 \cap Q2$ to prove the conclusion. QED

Outline

TLA$^+$ Specifications and Proofs

Size of the development (loc, without comments/blank lines)

	definitions	lemmas	invariant proof
specification	115	67	29
election invariant	11	54	158
log invariant	26	37	430
commit invariant	42	169	674
totals	194	327	1291

Summing Up

- High-level specification of Raft
  - understand the principles on which the algorithm is built
  - exhibit and prove an inductive invariant
  - facilitate the study of variations and extensions
- Main insights gained
  - overall correctness argument follows the structure of the original paper
  - importance of focusing on primary commits in the proof
  - certain actions (updating terms, secondary commits) need not be driven by leader
- Work in progress
  - refine the specification to better localize some actions
  - validate Raft implementations against the resulting specification