Modeling and Validation of Raft Using Maude

20180462 채승현

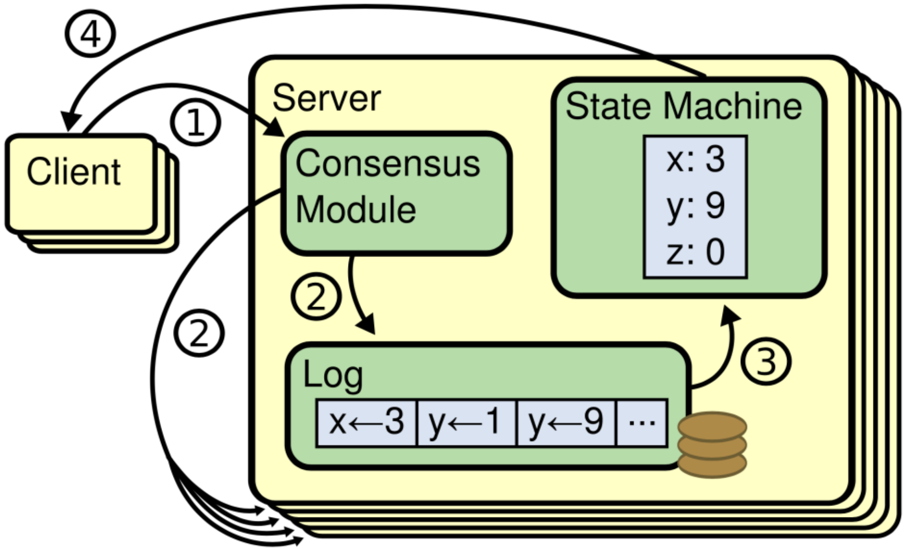
Table of Content

1.

Raft Algorithm

What is Consensus Algorithm?

Consensus Algorithm: Protocols to keep replicated logs consistent



Replicated state machine.
Adapted from *In Search of an Understandable Consensus Algorithm*, Ongaro, D. & Ousterhour, j.

What is Raft Algorithm?

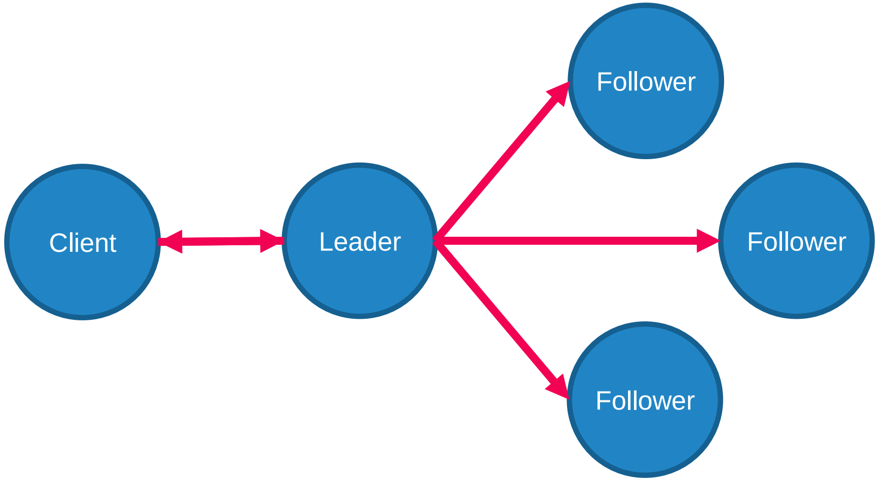
Raft: Consensus algorithm for managing a replicated log

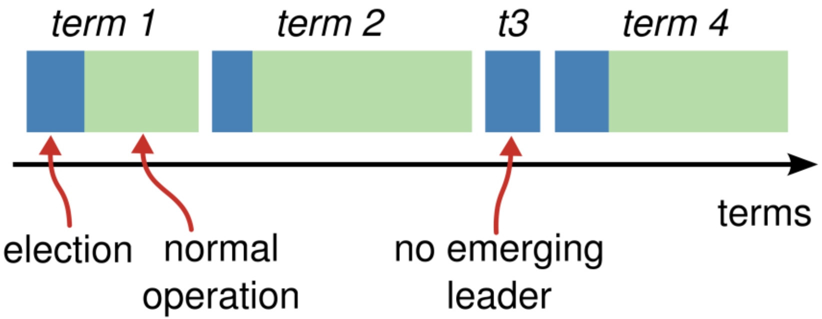
- Leader Election: a new leader must be chosen when an existing leader fails
- Log Replication: the leader must accept log entries from clients and replicate them across the cluster, forcing the other logs to agree with its own
- Safety

Raft Algorithm Basics

- At any given time each server is in one of three states: leader, follower, candidate

- Servers communicate using RPC (remote procedure calls)



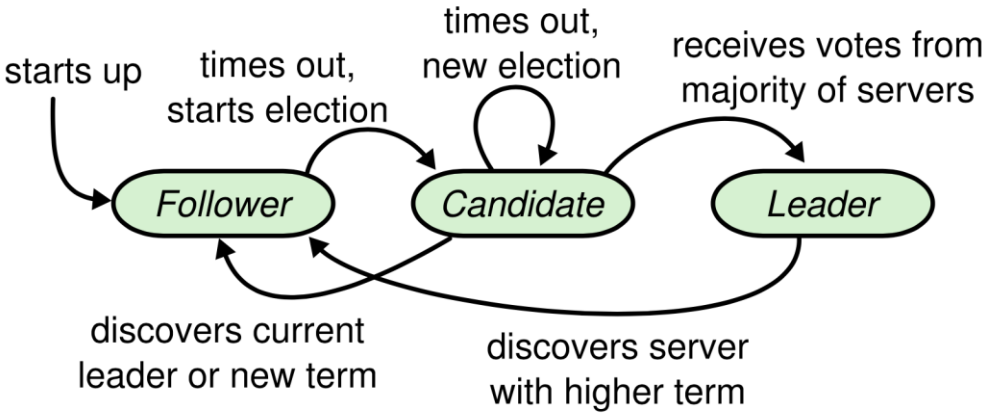


Time divided in unit of terms.
Adapted from *In Search of an Understandable Consensus Algorithm*, Ongaro, D. & Ousterhour, j.

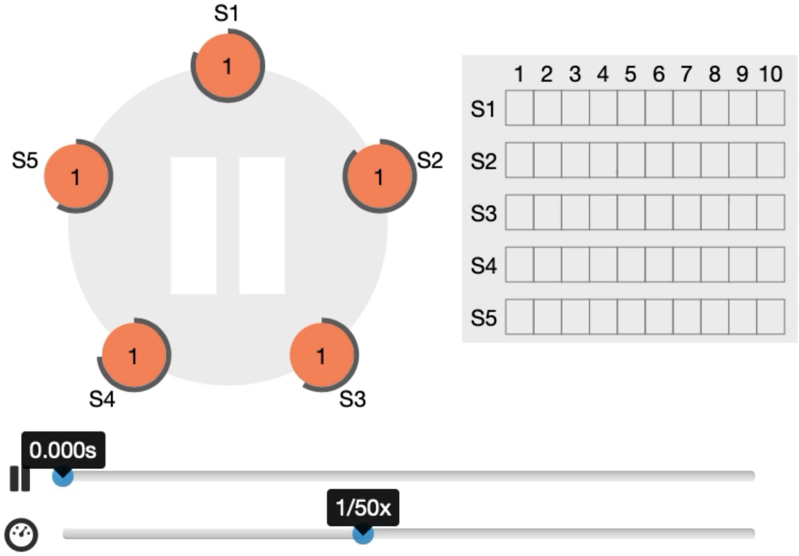
- Divides time into terms of arbitrary length

Raft Algorithm: Leader Election

A new leader must be chose when an existing leader fails.

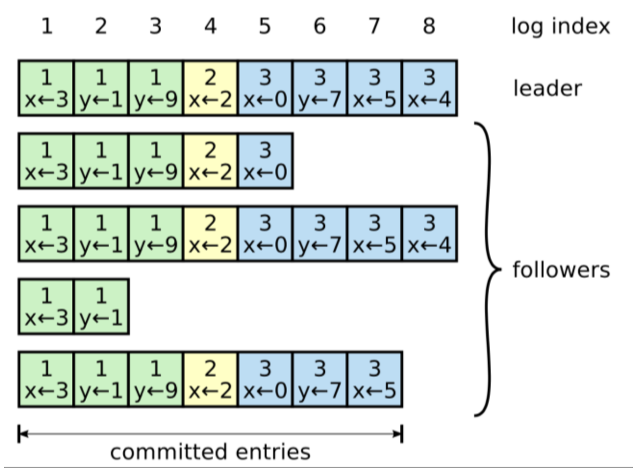


Server states
Adapted from *In Search of an Understandable Consensus Algorithm*, Ongaro, D. & Ousterhour, j.

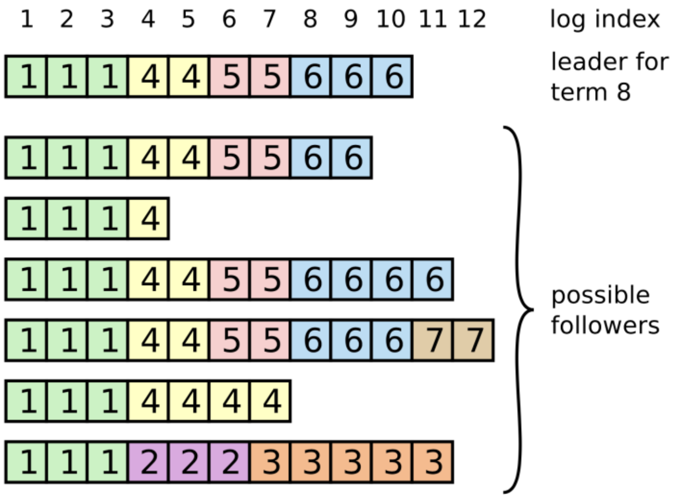


Raft Algorithm: Log Replication

The leader must accept log entries from clients and replicate them across the cluster, forcing the other logs to agree with its own.



Log comprised of entries
Adapted from *In Search of an Understandable Consensus Algorithm*, Ongaro, D. & Ousterhour, j.



Log inconsistencies
Adapted from *In Search of an Understandable Consensus Algorithm*, Ongaro, D. & Ousterhour, j.

2.

Modeling Using Maude

Simplification

Randomization and timing poses difficulties for Maude
→ Needs simplification

- Simplification: Removal of heartbeat system.
  - Ring shaped cluster structure
  - Automatic overwriting of followers’ log
- Offline server abstraction

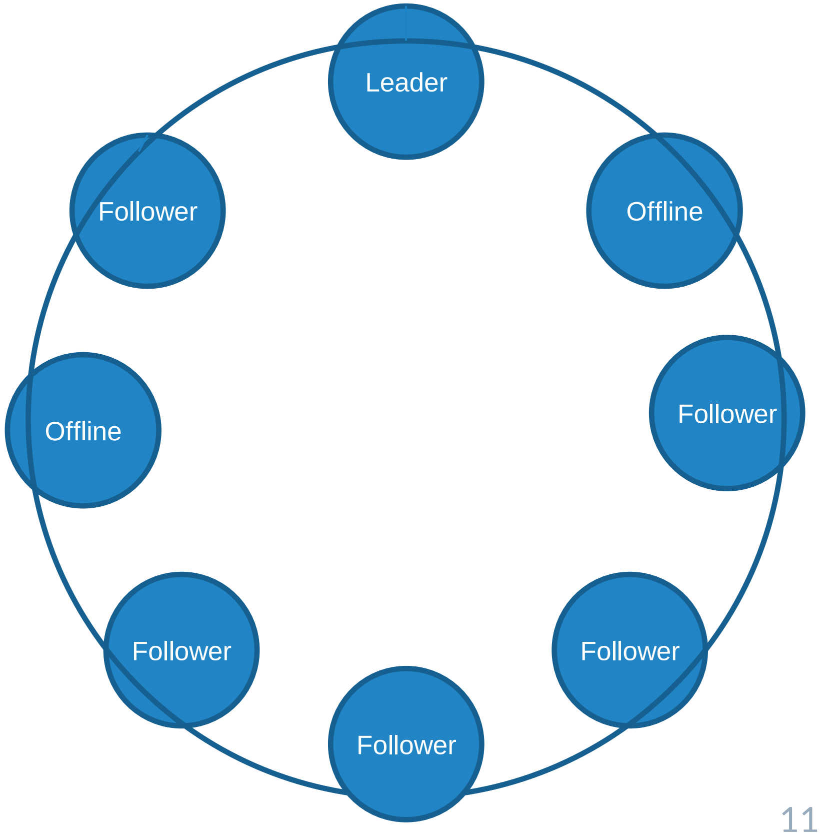
Simplification: Drawback Solutions

Ring shaped Cluster

: Instead of random candidate, next neighbor starts election (next neighbor election)

Automatic overwriting of followers' log

: When a leader adds entry, immediately sends entire log to followers to be set



Overall Model Structure

Comprised of 7 modules

- Node
  - Leader
  - Candidate
  - Follower
  - Offline

node.maude

- Log
- Entry

log.maude

- Node-Init
- Simul

raft.maude

- Message
- Multicast

message.maude

Node

- nodeType: current state of the node
- currentTerm: current term of the node
- log: current log
- Committed: committed entries
- Neighbors: all other nodes
- Waiting: Boolean flag indicating status of returning response
- Next-neighbor: next node in the ring structure

Follower

- become-leader: multicast new term and last entry to request vote

```
rl [become-leader] :
  (msg BecomeLeader(New-Term) from Lea to Fol)
  < Fol | nodeType : "Follower", currentTerm : Term     , log : Log ; E, neighbors : Fols, yes : N1, number-response : N2, AS >
=>
  < Fol | nodeType : "Candidate", currentTerm : New-Term, log : Log ; E, neighbors : Fols, yes : 1 , number-response : 0 , AS >
  (multicast RequestVote(New-Term, E) from Fol to Fols) .
```

- append-entry-follower: set log to that of leader's

```
crl [append-entry-follower] :
  (msg AppendEntry(New-Term, Log) from Lea to Fol)
  < Fol | nodeType : "Follower", currentTerm : Term     , log : OldLog, AS >
=>
  < Fol | nodeType : "Follower", currentTerm : New-Term, log : Log,     AS >
  (msg AppendEntryResponse(New-Term, true) from Fol to Lea)
  if New-Term >= Term .
```

Candidate

- receive-yes-vote: receive yes votes from followers

```
crl [receive-yes-vote] :
  (msg Vote(Term, B) from Fol to Lea)
  < Lea | nodeType : "Candidate", currentTerm : Term, yes : N1    , number-response : N2    , AS >
=>
  < Lea | nodeType : "Candidate", currentTerm : Term, yes : N1 + 1, number-response : N2 + 1, AS >
  if B == true .
```

- voted-leader: Become leader and propagate log if majority voted yes

```
crl [voted-leader] :
  < Lea | nodeType : "Candidate", currentTerm : Term, log : Log, neighbors : Fols, waiting : B   , majority : N1, yes : N2, number-neighbors : N3, number-response : N3, AS >
=>
  < Lea | nodeType : "Leader", currentTerm : Term, log : Log, neighbors : Fols, waiting : true, majority : N1, yes : 1 , number-neighbors : N3, number-response : 0 , AS >
  (multicast AppendEntry(Term, Log) from Lea to Fols)
  if N2 >= N1 .
```

Leader

- request-leader: for client request, add to log and send to followers

```
crl [request-leader] :
  (msg Request(C) from Cli to Lea)
  < Lea | nodeType : "Leader", currentTerm : Term, log : Log , neighbors : Fols, waiting : false, yes : N1, number-response : N2, AS >
=>
  < Lea | nodeType : "Leader", currentTerm : Term, log : Log', neighbors : Fols, waiting : true , yes : 1 , number-response : 0 , AS >
  (multicast AppendEntry(Term, Log') from Lea to Fols)
  if Log' := Log ; entry(index(head(Log)) + 1, Term, C) .
```

- commit: if entry appended for majority, commit to state machine

```
crl [commit] :
  < Lea | nodeType : "Leader", currentTerm : Term, waiting : true , neighbors : Oids, log : Log, committed : ComLog, majority : N1, yes : N2, number-neighbors : N3, number-response : N3, AS >
=>
  < Lea | nodeType : "Leader", currentTerm : Term, waiting : false, neighbors : Oids, log : Log, committed : Log   , majority : N1, yes : N2, number-neighbors : N3, number-response : N3, AS >
  (multicast Commit(Term, head(Log)) from Lea to Oids)
  if N2 >= N1 .
```

Offline

- leader-offline: next neighbor to start election

```
rl [leader-offline] :
  (msg BecomeLeader(Term) from Lea to O1)
  < O1 | nodeType : "Offline, next-neighbor : O2, AS >
=>
  < O1 | nodeType : "Offline, next-neighbor : O2, AS >
  (msg BecomeLeader(Term) from O1 to O2) .
```

- request-vote-offline: vote no

```
rl [request-vote-offline] :
  (msg RequestVote(Term, E) from Lea to Fol)
  < Fol | nodeType : "Offline, AS >
=>
  < Fol | nodeType : "Offline, AS >
  (msg Vote(Term, false) from Fol to Lea) .
```

3.
Validation Using Maude

Properties of Raft

Election Safety

At most one leader can be elected in a given term

Leader Append-Only

A leader never overwrites or deletes entries in it log, it only appends new entries

Log Matching

If two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index

Leader Completeness

If a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms

State Machine Safety

If a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index

1. Election Safety

At most one leader can be elected in a given term

$$\forall\ e, f \in elections:$$
$$e.eterm = f.eterm \Rightarrow e.eleader = f.eleader$$

```
Maude> search [1] init(3, 1) =>* C:Configuration
> < O1:Oid : LeaderNode | AS1:AttributeSet >
> < O2:Oid : LeaderNode | AS2:AttributeSet > .
search [1] in SIMUL : init(3, 1) =>* C:Configuration < O2:Oid : LeaderNode | AS2:AttributeSet >
< O1:Oid : LeaderNode | AS1:AttributeSet > .

No solution.
states: 2596992  rewrites: 854539870 in 12674940ms cpu (12674940ms real) (67419 rewrites/second)
```

2. Log Matching

If two logs are the same at an arbitrary index and term, all logs prior are equal

$$\begin{aligned}&\forall\, l, m \in allLogs : \\ &\quad \forall\, \langle index, term\rangle \in l : \\ &\qquad \langle index, term\rangle \in m \Rightarrow \\ &\qquad\quad \forall\, pindex \in 1..index : \\ &\qquad\qquad l[pindex] = m[pindex]\end{aligned}$$

```
Maude> search [1] init(3, 1) =>* C:Configuration
> < O1:Oid : C1:Cid | log : L11:Log ; entry(ind:Nat, term:Nat, C1:Command) ; L12:Log, AS1:AttributeSet >
> < O2:Oid : C2:Cid | log : L21:Log ; entry(ind:Nat, term:Nat, C2:Command) ; L22:Log, AS2:AttributeSet >
> such that C1:Command =/= C2:Command or L11:Log =/= L21:Log .
search [1] in SIMUL : init(3, 1) =>* C:Configuration < O2:Oid : C2:Cid | AS2:AttributeSet,log : (L21:Log ; entry(ind:Nat,
 term:Nat, C2:Command) ; L22:Log) > < O1:Oid : C1:Cid | AS1:AttributeSet,log : (L11:Log ;
    entry(ind:Nat, term:Nat, C1:Command) ; L12:Log) > such that C1:Command =/= C2:Command or L11:Log =/= L21:Log = true .

No solution.
states: 2596992  rewrites: 1072013854 in 15018100ms cpu (15018105ms real) (71381 rewrites/second)
```

3. State Machine Safety

If a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index

$$
\begin{aligned}
&\forall\, i \in \mathit{Server} : \\
&\quad \land\ \mathit{commitIndex}[i] \leq \mathit{Len}(\mathit{log}[i]) \\
&\quad \land\ \forall\, \langle \mathit{index}, \mathit{term} \rangle \in \mathit{log}[i] : \\
&\qquad\qquad \mathit{index} \leq \mathit{commitIndex}[i] \Rightarrow \\
&\qquad\qquad\quad \langle \mathit{index}, \mathit{term} \rangle \in \mathit{committed}(\mathit{currentTerm}[i])
\end{aligned}
$$

```
Maude> search [1] init(3, 1) =>* C:Configuration
> < O1:Oid : C1:Cid | committed : L11:Log ; entry(ind:Nat, term:Nat, C1:Command) ; L12:Log, AS1:AttributeSet >
> < O2:Oid : C2:Cid | committed : L21:Log ; entry(ind:Nat, term:Nat, C2:Command) ; L22:Log, AS2:AttributeSet >
> such that C1:Command =/= C2:Command or L11:Log =/= L21:Log .
search [1] in SIMUL : init(3, 1) =>* C:Configuration < O2:Oid : C2:Cid | AS2:AttributeSet,committed : (L21:Log
; entry(ind:Nat, term:Nat, C2:Command) ; L22:Log) > < O1:Oid : C1:Cid | AS1:AttributeSet,
    committed : (L11:Log ; entry(ind:Nat, term:Nat, C1:Command) ; L12:Log) > such that C1:Command =/= C2:Comman
d or L11:Log =/= L21:Log = true .

No solution.
```

Conclusion

- Studied Raft Algorithm
- Modeled Raft based on pre-existing work
- Validates properties using search command

- Use Probabilistic rewrite rules to model time and randomization
- Use init(5, 2) initial state
- LTL model checker

References

- Ongaro, D. & Ousterhout, J. In Search of an Understandable Consensus Algorithm, USENIX, 2014
- Ongaro, D. Consensus: Bridging Theory and Practice. PhD thesis, Standford University, 2014
- Stephens, S. From Models to Implementations – Distributed Algorithms using Maude, B.S. thesis, UIUC, 2018

Thank You for Listening