Planning for Change in a Formal Verification of the Raft Consensus Protocol

Doug Woos  James Wilcox  Steve Anton  Zach Tatlock  Mike Ernst  Tom Anderson
Contributions

First formal proof of Raft’s safety

first verified implementation!

Large-scale Verdi case study

stress test; reverification inevitable

Proof engineering lessons

affinity lemmas, etc.
Distributed Systems
Reliably deliver procrastination
Also serious infrastructure
One day last summer...

The New York Times
The Stock Market Bell Rings, Computers Fail, Wall Street Cringes

By NATHANIEL POPPER  JULY 8, 2015

Problems with technology have at times roiled global financial markets, but the 223-year-old New York Stock Exchange has held itself up as an oasis of humans ready to step in when the computers go haywire.

On Wednesday, however, those working on the trading floor were left helpless when the computer systems at the exchange went down for nearly four hours in the middle of the day, bringing an icon of capitalism’s ceaseless energy to a costly halt.

The exchange ultimately returned to action shortly before the closing bell,
One day last summer...
One day last summer...
How distributed systems fail
Related Work

EventML [LADA12, AVoCS15]
language for verified distributed systems

IronFleet [SOSP15]
liveness, log compaction, serialization

Verdi [PLDI15]
network semantics, transformers, higher-order
Verdi background

Network semantics
*operational semantics define network behavior*

Verified system transformers
*prove property transfer to adversarial network*

∀ Φ S, holds(Φ, S, \text{\textasciitilde}₁) →
holds(transfer(Φ), T(S), \text{\textasciitilde}₂)
Big Picture

Past: Verdi Framework

*compositional fault tolerance*

Present: Verified Raft

*critical piece of infrastructure*

Future:

*dynamically upgrading systems program logic*
Outline

Verification Challenge

*state machine replication*

Raft Algorithm

*implemented in Verdi*

Proof Overview

*and lessons learned*
Replication for fault tolerance

critical components must not fail
Replication for fault tolerance

replicas must be consistent with each other

available if n/2 nodes are up
Replication for fault tolerance
Replication correctness
Replication correctness

- Linearizability
- Cluster presents consistent order of operations to clients
Internal Correctness

linearizability follows from internal correctness: state machine safety
Goal: Verify Raft

Prove State Machine Safety
Reduce linearizability to State Machine Safety [PLDI15]
Goal: Verify Raft
Outline

Verification Challenge
state machine replication

Raft Algorithm
implemented in Verdi

Proof Overview
and lessons learned
Formalizing the network

\[(P, \Sigma, T)\]

- packets in flight
- history of I/O
- data @ nodes
- state of the world
Formalizing the network

$(P, \Sigma, T)$
Formalizing the network

$$(P, \Sigma, T) \sim (P', \Sigma', T')$$
Defining network semantics

\[
H_{\text{net}}(\text{dst}, \Sigma[\text{dst}], \text{src}, m) = (\sigma', o, P') \quad \Sigma' = \Sigma[\text{dst} \mapsto \sigma']
\]

\[
\frac{(\{(\text{src}, \text{dst}, m)\} \uplus P, \Sigma, T) \leadsto (P \uplus P', \Sigma', T ++ \langle o \rangle)}{\text{DELIVER}}
\]
Defining network semantics

\[
\begin{align*}
H_{\text{net}}(\text{dst}, \Sigma[\text{dst}], \text{src}, m) &= (\sigma', o, P') \\
\Sigma' &= \Sigma[\text{dst} \mapsto \sigma'] \\
((\text{src}, \text{dst}, m) \cup P, \Sigma, T) &\rightsquigarrow (P \cup P', \Sigma', T ++ \langle o \rangle)
\end{align*}
\]

**Deliver**

\[
p \in P \\
(P, \Sigma, T) \rightsquigarrow (P \cup \{p\}, \Sigma, T)
\]

**Duplicate**

\[
\{p\} \cup P, \Sigma, T) \rightsquigarrow (P, \Sigma, T)
\]

**Drop**

\[
H_{\text{tmt}}(n, \Sigma[n]) = (\sigma', o, P') \\
\Sigma' = \Sigma[n \mapsto \sigma'] \\
(P, \Sigma, T) \rightsquigarrow (P \cup P', \Sigma', T ++ \langle \text{tmt}, o \rangle)
\]

**Timeout**
Defining network semantics

\[
H_{\text{net}}(\text{dst}, \Sigma[\text{dst}], \text{src}, m) = (\sigma', o, P') \quad \Sigma' = \Sigma[\text{dst} \rightarrow \sigma']
\]

\[
\left\{ (\text{src}, \text{dst}, m) \right\} \rightarrow (P, \Sigma, T) \rightsquigarrow (P \oplus P', \Sigma', T + \langle o \rangle)
\]

\[
H_{\text{tmt}}(n, \Sigma[n]) = (\sigma', o, P') \quad \Sigma' = \Sigma[n \rightarrow \sigma']
\]

\[
(P, \Sigma, T) \rightsquigarrow (P \oplus P', \Sigma', T + \langle \text{tmt, o} \rangle)
\]
Implementing Raft

Term 1

Term 2

Term 3

...
Implementing Raft: Leader Election

Term 1

Term 2

Term 3

Candidates: Request Vote

Followers: Vote

...
Implementing Raft
Implementing Raft: Log Replication

Leader commits entry when receiving \( n/2 \) acks
Outline

Verification Challenge
state machine replication

Raft Algorithm
implemented in Verdi

Proof Overview
and lessons learned
Verifying Raft: Show linearizability
Verifying Raft: Approach
State Machine Safety

Nodes agree about committed entries

since only committed entries executed

proof by induction on an execution
State Machine Safety: Proof

not inductive!
State Machine Safety: Proof

90 invariants in total

Lemma
Lemma
Lemma

I
true initially

I
preserved

I

I
The burden of proof

Re-verification is the primary challenge:
- invariants are not inductive
- not-yet-verified code is wrong
- need additional invariants
The burden of proof

Re-verification is the primary challenge

Proof engineering techniques help:
- affinity lemmas
- intermediate reachability
- structural tactics
- information hiding
Ghost State: Example

Capture all entries received by a node

- Follower
  - Append: [A], B, C
- Leader
  - Log (real): A, B, C
  - allEntries (ghost): \{A, B, C\}
Affinity Lemmas: Example

\[ e \in \text{log} \implies e.\text{term} > 0 \]

\[ e \in \text{allEntries} \implies e.\text{term} > 0 \]
Affinity Lemmas: Example

\[ e \in \text{log} \implies P \ e \]

\[ e \in \text{allEntries} \implies P \ e \]

every invariant of entries in logs is invariant of entries in allEntries
Affinity Lemmas

Ex 1: Relate ghost state to real state
	transfer properties once and for all

Ex 2: Relate current messages to past
	respone => past request
Structured Handlers: Example

\[
\text{handler} = \text{update\_state} \; ; \; \text{respond}
\]

1. \(\text{net} \xrightarrow{\text{handler}} \text{net}'\)
2. \(\text{net} \xrightarrow{\text{update\_state}} \text{net}_i\)
3. \(\text{net}_i \xrightarrow{\text{respond}} \text{net}'\)
Structured Handlers: Example

handler = update_state ; respond

handler

I

net

I

net’

I

update_state

net

net_i

respond

I

net’
Structured Handlers: Example

```
handler = update_state ; respond
```

```
handler
  /\net
 /   \net'
     /\net_i
    /  \update_state
     /   \respond
     /    \net'
```
The burden of proof

Re-verification is the primary challenge

Proof engineering techniques help:
- affinity lemmas
- intermediate reachability
- structural tactics
- information hiding
Contributions

First formal proof of Raft’s safety

*first verified implementation!*

Large-scale Verdi case study

*stress test; reverification inevitable*

Proof engineering lessons

*affinity lemmas, etc.*
Planning for Change in a Formal Verification of the Raft Consensus Protocol

Doug Woos  James Wilcox  Steve Anton  Zach Tatlock  Michael Ernst  Tom Anderson

UNIVERSITY of WASHINGTON