Formal Modeling of Distributed Systems

Modeling the Raft Distributed Consensus Protocol in LNT

Hugues Evrard - Google
MARS’22
Munich, 2 April 2022
whoami

- Hugues ("Hugh") Evrard
- PhD: Team Convecs, Inria Grenoble, France
- PostDoc: Imperial College London, UK
- GraphicsFuzz startup
- Google (London, now Paris)

Opinions are my own and not the views of my employer
Agenda

- Distributed consensus
- Raft protocol
- Modeling with LNT
- Modeling distributed systems
A crash-proof service

[Schneider-90] Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial
A crash-proof service

[Schneider-90] *Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial*
A crash-proof service

[Schneider-90] Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial

apply the same cmds list
A crash-proof service

[Schneider-90] *Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial*
Distributed consensus

- Distributed: several nodes
- Nodes may crash
- Nodes communicate via asynchronous messages
- Unreliable channels: messages can be dropped, duplicated, reordered

Consensus: can nodes agree on something?
Distributed consensus

- Distributed: several nodes
- Nodes may crash
- Nodes communicate via asynchronous messages
- Unreliable channels: messages can be dropped, duplicated, reordered

Consensus: can nodes agree on something?

If using a deterministic protocol, then it’s impossible (FLP impossibility)

[Fischer-Lynch-Paterson-85] Impossibility of distributed consensus with one faulty process
Distributed consensus: (non-deterministic) protocols

- Paxos: [Lamport-90-98] *The Part-Time Parliament*
- Rich literature:
  - Multi-Paxos
  - “Paxos made easy”
  - …
- Raft: [Ongaro-Ousterhout-13] *In Search of an Understandable Consensus Algorithm*
  - [https://raft.github.io/](https://raft.github.io/)
  - Focus on *clarity* and *understandability*
  - Specification in TLA
  - Manual proof
Raft in a nutshell (1)

- Time divided in terms
- At each term:
  - 1. Leader election: elect one leader among nodes
Raft in a nutshell (2)

- Time divided in **terms**
- At each term:
  - 1. **Leader election**: elect one leader among nodes
  - 2. **Append log entries**: leader replicates log entries to quorum of followers

```
node1
  leader

node2
  cmd1
  cmd2

node3
  cmd1
```
Raft in a nutshell (2)

- Time divided in **terms**
- At each term:
  - 1. **Leader election**: elect one leader among nodes
  - 2. **Append log entries**: leader replicates log entries to quorum of followers

```
client  ➔ cmd3
  ➔ node1
    ➔ cmd3, 2
      ➔ ok
      ➔ cmd3, 2
      ➔ no, 1

node1
  ➔ leader
  ➔ cmd1
  ➔ cmd2
  ➔ cmd3

node2
  ➔ cmd1
  ➔ cmd2
  ➔ cmd3

node3
  ➔ cmd1

commit index: #cmds replicated to majority of nodes
```
Raft in a nutshell (2)

- Time divided in **terms**
- At each term:
  - 1. **Leader election**: elect one leader among nodes
  - 2. **Append log entries**: leader replicates log entries to quorum of followers

![Diagram]

```
client

resp3

node1

cmd1

cmd2

cmd3

node2

node3

leader

cmd1

cmd2

cmd3

commit index: #cmds replicated to majority of nodes
```
Raft in a nutshell (3)

- Time divided in **terms**
- At each term:
  - 1. **Leader election**: elect one leader among nodes
  - 2. **Append log entries**: leader replicates log entries to quorum of followers

- Only the leader interacts with the client
- Any node can timeout and start a new election
- Leader sends heartbeat messages to prevent timeouts

- Used in the industry
  - e.g. Hashicorp’s Consul, etcd (Kubernetes)
Verification? Start with **formal model**
Formal modeling with LNT: a primer

module primer is

channel CalcOp is
  (op1, op2, res: nat)
end channel

process Calc [Add, Mul: CalcOp] is
  var op1, op2, res: nat in
  loop
    select
      Add(?op1, ?op2, ?res)
      where res == (op1 + op2)
    []
      Mul(?op1, ?op2, ?res)
      where res == (op1 * op2)
    end select
  end loop
end var
end process

process User [Op: CalcOp] (a, b: nat) is
  var result: nat in
  Op(a, b, ?result)
end var
end process

process Main [Add, Mul: CalcOp] is
  par
    Add -> User[Add](1, 2)
    ||
    Mul -> User[Mul](3, 4)
    ||
    Add, Mul -> Calc[Add, Mul]
  end par
end process
end module

State space as LTS: Labelled Transition System
Modeling Raft in LNT (1): top-level parallel composition

```
par Send, Recv in
  par
    Node
  ||
    Node
  ||
    Node
  ||
    ...
end par
||
Network
end par
```

replicated server nodes
process Network is
    var bag: MsgSet := {} in
    loop
        select
            Send(?msg);    \{ good reception \}
            bag := insert(msg, bag)  \}
            Send(?msg);  \{ msg lost (not stored in bag) \}
            []
            Recv(msg) where member(msg, bag);  \{ transmit, possible reordering \}
            bag := remove(msg, bag)
            []
            Recv(msg) where member(msg, bag);  \{ msg duplication: transmit & keep in bag \}
        end select
    end loop
end var
end process
Modeling Raft in LNT (3): Node with Crash in select

process Node is
  (* init ... *)
  loop
    select
      Recv(?msg);
      case msg in
        VoteRequest -> ... Send(msg) ... |
        AppendEntries -> ... Send(msg) ... |
      end case
[[]]
    Timeout;
    (* start election or send heartbeat *)
[[]]
    Client(?cmd) where state == Leader;
    (* add client command in local log *)
[[]]
    Crash;
    break
  end select
end loop
end process
process Node is
(* init ... *)
loop
select
    Recv(?msg);
  case msg in
    VoteRequest -> ... Send(msg) ...
| AppendEntries -> ... Send(msg) ...
end case
[]
Timeout;
(* start election or send heartbeat *)
[]
Client(?cmd) where state == Leader;
(* add client command in local log *)
[]
Crash;
break
end select
end loop
end process

Simplified LTS

Event, reaction

Cannot crash here?
Modeling Raft in LNT (4): Node with Crash in disrupt

process Node is
    (* init ... *)
    disrupt
    loop
        select
           Recv(?msg);
            case msg in
                VoteRequest -> ... Send(msg) ...
                | AppendEntries -> ... Send(msg) ...
            end case
        []
            Timeout;
            (* start election or send heartbeat *)
        []
        ... 
        end select
    end loop
    by
        Crash
    end disrupt
end process
Issues found in the original TLA specification

- Typo-style error
  - missing apostrophe denoting future state
- Missing node state transition: candidate did not step down
  - Different from the behavior described in plain English in the paper
  - Did not jeopardize the manual proof

- Both have been fixed since.
- Pretty hard to get a spec right!
Modeling distributed systems
LNT / CADP formal development environment

- Writing a formal specification ~= writing a program
- Want a **quick feedback loop**
  - like REPL or fast edit-compile-run cycles
- LNT + CADP offers:
  - LNT: Mainstream programming language syntax
  - Strong typing, good error messages
  - Very powerful parallel composition and inter-process communication
    [Garavel-Serwe-17] *The Unheralded Value of the Multiway Rendezvous*
  - Fast compile time
  - “assert” keyword to fail early at state-space generation time
    - debug: can still inspect the state space generated so far
  - generate *implicit* state space
    - manual step-by-step exploration to inspect/debug the spec
Generic models for distributed systems

process Node is
    (* init local state ... *)
    disrupt
    loop
        select

            Recv(?msg);
            (* update local state, send messages *)

            []

            LocalEvent; (* e.g. timeout, read sensor, ... *)
            (* update local state, send messages *)

        end select
    end loop
by
    Crash (* local failure *)
end disrupt
end process

generic skeleton
(like Erlang’s gen_server)
A library of network models

- Network oblivious to protocol details
  - Just transfer messages
- Can write various network semantics
  - synchronous/asynchronous
  - drop message, or not
  - reorder message, or not
- Can switch between network modules with no change anywhere else in the spec!

```vhdl
(* Transfer any message immediately *)
process ReliableSynchronousNetwork is
  loop
    Send(?msg);
    Recv(msg)
  end loop
end process
```
Modeling choices

- Want to keep state space size under control
- Model only the necessary, but no less
  - Distributed systems: inter-node communication, outstanding local events
  - Hide the rest as much as feasible
- Some examples in our Raft model:
  - A candidate directly votes for itself, rather than sending itself a vote request
  - Do not respond to stale RPC requests
    - The TLA spec does, to promptly inform a node that it is outdated
  - Append only one entry at a time (i.e. do not batch entries)
  - Force the order in which VoteRequests are broadcasted
    - Rely on network semantics to model reordering
Most distributed protocols are robust to message duplication:
  - Have idempotent messages
  - Receive it once, then drop duplicates
Assume this robustness: no need to model message duplication
This can typically save a lot of state space!
Modeling shortcuts: watchout for pitfalls!

- Taking shortcuts in modeling is a **very slippery slope**!
- It is **very easy** to make wrong assumptions there
  - Better be safe than sorry!
You’ve got a verified model, now what?
Implement. And introduce bugs 😞
Direct formal-model-to-implementation approaches
- [Evrard-15] Distributed LNT Compiler: LNT to distributed C with TCP sockets
- [deMoura-et-al-15] Lean: both theorem prover & compiler to Javascript
- ... 
Need good tooling
- debugger, profiler, package manager, etc
Wild request: next gen language’s specification is formally defined
- Avoid/reduce undefined behaviors
- Sane basis for FM: stop reverse-eng/afterthought FM once the language is out!
- Also formalize the ISAs (See e.g. Alastair Reid’s work on ARM ISA)
You've got a verified model, now what?

Implement. And introduce bugs 😞

Direct formal-model-to-implementation approaches

- [Evrard-15] Distributed LNT Compiler: LNT to distributed C with TCP sockets
- [deMoura-et-al-15] Lean: both theorem prover & compiler to Javascript

Need good tooling

- debugger, profiler, package manager, etc

Wild request: next gen language’s specification is formally defined

- Avoid/reduce undefined behaviors
- Sane basis for FM: stop reverse-eng/afterthought FM once the language is out!
- Also formalize the ISAs (See e.g. Alastair Reid’s work on ARM ISA)

Quoting https://doc.rust-lang.org/reference/:

Finally, this book is not normative. It may include details that are specific to rustc itself, and **should not be taken as a specification for the Rust language**. We intend to produce such a book someday, and until then, the reference is the closest thing we have to one.

- Sane basis for FM: stop reverse-eng/afterthought FM once the language is out!
- Also formalize the ISAs (See e.g. Alastair Reid’s work on ARM ISA)
Conclusion

- Modeling Raft in LNT
- Formal modeling of distributed systems in general
- Modeling approaches to keep state space size small
  - Powerful, but watch out for semantics pitfalls!
- Bridge the gap with between formal models and implementation
Thanks!

- Questions?
Formal methods at Google? Some examples:

- pKVM (Android Hypervisor): formal semantics of ARM-v8a, see e.g. Peter Sewell’s recent papers
- OpenTitan: code verified via Dafny [https://github.com/lowRISC/opentitan/pull/10143](https://github.com/lowRISC/opentitan/pull/10143)
- BoringSSL has code verified via Fiat (MIT) [https://boringssl.googlesource.com/boringssl/+refs/heads/master/crypto/curve25519/curve25519.c#2015](https://boringssl.googlesource.com/boringssl/+refs/heads/master/crypto/curve25519/curve25519.c#2015)