Programming abstractions for automating the verification of replicated state machine

Cezara Drăgoi, INRIA Paris/ENS

joint work with
Tom Henzinger, IST Austria
Josef Widder, TU Wien
Damien Zufferey, MPI Kaiserslautern
Replication

![Diagram showing a crash and a replicated system]

**Replication**
Replicated state machine

Zab
Viewstamped

Generalized Paxos
Goal

Design **automated verification** for **implementations** of replicated state machine

Difficulties

- **Impossibility result** [FLP’85]: “Consensus cannot be reached in asynchronous networks in the presence of at least one faulty process”
- protocols have been developed for **different network assumptions**: Different degrees of synchrony and faults
Difficulties: Complex control structure

- asynchronous parallel composition
Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passing: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message? what if two messages are received in the reverse sending order?
Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message? what if two messages are received in the reverse order?
- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.
Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message? what if two messages are received in the reverse order?
- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.
- faults: message drop, process-crash, messages are corrupted
Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message? what if two messages are received in the reverse order?
- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.
- faults: message drop, process-crash, messages are corrupted
Mechanized verification:
- Verdi, EventML(2012) (only safety)
- IronFleet(2013)
- TLA+

Automated verification:
- model checking of simple algorithms or protocols using minor coordination
- Ivy(2016)
- Psync
Goal

Programming abstraction
simpler source code
+ specifications

Automated Verification
Assumptions: synchronous communication and no faults

there is a bound $b$ on the message delay

Assumptions: the leader receives the messages from all processes and every process hears from the leader
Example: Communication

every process hears from the leader

Change leader

Wait longer

Heartbeats

Propose/Commit
Example: Communication

- **Propose**: A
- **Acknowledge**: A
- **Commit**: A

**the leader hears from all processes**

- **Wait longer**
- **Heartbeats**
- **Change leader**
- **Acknowledgment**
- **Commit**

- **Acknowledgment**
We would like a round based model that separates the network properties from the algorithmic computation.

- The leader hears from all processes,
- every process hears from the leader,
- compute the minimum value,
- a common prefix, etc.
Programming model

1. Separates the *network properties* from the *algorithmic computation*

2. Faults and asynchrony simulated by an adversarial environment

3. We want a *simple programming abstraction* that offers a synchronous image of the system and compiles into *executable asynchronous code*. 
A **round** is a computation step that
- defines *the interaction between the processes* that participate in that step,
- processes synchronize at the boundary between rounds,
- it gives a logical unit of time.

**Communication close** = two rounds do not communicate with each other, i.e., messages are scoped in the round.
A round is a computation step that
• defines *the interaction between the processes* that participate in that step,
• processes synchronize at the boundary between rounds,
• it gives a logical unit of time.

Communication close = two rounds do not communicate with each other, i.e., messages are scoped in the round.
HO-model [Charrone-Bost, Schiper'09]
Each process $p$ has a variable, $HO(p)$: the set of processes $p$ hears-from $HO(p)$ is non-deterministically chosen by an adversarial environment.

1. Determines the delivered messages
Each process \( p \) has a variable, \( \text{HO}(p) = \) the set of processes \( p \) hears-from.

\( \text{HO}(p) \) is non-deterministically chosen by an adversarial environment.

1. Determines the delivered messages
2. The network assumptions are stated over the HO variables

\[ \forall p. p \in \text{HO(leader)} \land \text{leader} \notin \text{HO}(p) \]
\( \square \Diamond (\forall p. p \in HO(\text{Leader}) \land \text{Leader} \in HO(p)) \)

\( \square (\forall p. p \in HO(\text{leader}) \land \text{leader} \in HO(p)) \)
Approach

Replicated state machine implementations \rightarrow HO-computational model \rightarrow Psync \rightarrow Runtime \rightarrow Replicated state machine implementations \rightarrow Verifier \rightarrow proof or counterexample
PSync Program Structure \[\text{popl}'16\]

- **Program**
  - Interface
  - Variable
  - Init
  - Round_\(T\)

- **Round_\(T\)**
  - Send: \((\cdot) \to [Id \to T]\)
  - Comm Pred
  - Update: \([Id \to T] \to (\cdot)\)
Communication Predicates:

Zab, Viewstamped \( \Box (\forall p. p \in \text{HO}(\text{Leader}) \land \text{Leader} \in \text{HO}(p)) \)

Paxos \( \Box \Diamond (\forall p. p \in \text{HO}(\text{Leader}) \land \text{Leader} \in \text{HO}(p)) \)
Example: Last Voting Algorithm

```
new Round[(Int, Time)]{
  def send(): Map[ProcessID, (Int, Time)] = Map( leader -> (x, ts) )
  def update(mailbox: Map[ProcessID, (Int, Time)]) {
    if (id == leader && mailbox.size > n/2) {
      vote = mailbox.maxBy(_._2._2)._2._1 // value with maximal ts
      commit = true
    }
  }
}
```
Outline

Psync

Runtime

Verifier

Replicated state machine implementations

Ⅰ= proof or counterexample
Discard late messages
Send
Receive
CommPred implementation
Accumulate
Update
Next round
Catching up
Runtime: Round switch
CommPred implementation
Partial synchrony

Network

Asynchronous | Synchronous | Asynchronous

Runtime Execution

Asynchronous | Asynchronous | Synchronous

Respects liveness assumptions

Next round

Send

Accum

TO

Update

Receive

Discard

Catching
Runtime Correctness

For any Psync program $P$, the runtime semantics of $P$ observationally refines the HO semantics of $P$, if the client is commutative.

$$\text{Clients} \sqsubset \text{Runtime}(P) \subseteq \text{Clients} \sqsubset \text{HO}(P)$$
Outline

Psync

Runtime

Verifier

Replicated state machine implementations

\( I = \) proof or counterexample
Psync: Benefits for Verification

Reason about rounds in isolation. Lockstep semantics, no interleaving.

Propose

Propose

Acknowledge

Acknowledge

Simple invariants that reason at the round boundaries, no messages are in flight, only the local states matter.
Psync: semi-automated verification

Psync program annotated with inductive invariants.

Specification

Formulas defining the inductiveness checks

∀ \( x^{old}, x^{new} \).
INV(\( x^{old} \)) \land TRound(\( x^{old}, x^{new} \))

INV(\( x^{new} \))

Verification Condition Generator

Expert

Decision procedure

CORRECT

YES

NO

Counter-example or ?
Hoare-style Verification [vmcai'14]

Init ; ( EnvHO ; Round)* ; ( EnvHO ; Round) ; ( EnvHO ; Round )*
Invariant for Agreement

\[ \forall i. \neg \text{decided}(i) \land \neg \text{ready}(i) \land \exists v, t, A. A = \{ i. \ ts(i) > t \} \land |A| > n/2 \]

\[ \land \quad \forall i. i \in A \Rightarrow x(i) = v \]

\[ \land \quad \forall i. \text{decided}(i) \Rightarrow x(i) = v \]

\[ \land \quad \forall i. \text{commit}(i) \lor \text{ready}(i) \Rightarrow vote(i) = v \]

\[ \land \quad t \leq \Phi \]

\[ \land \quad \forall i. ts(i) = \Phi \Rightarrow commit(coord) \]
Specification logic

• is able to express:
  -- properties of sets of processes in the network
  -- cardinality constraints
  -- properties of the data values stored by each process
  -- $\exists \forall$ quantifier alternation

• captures the transition relation of algorithms in the HO-model

• has a semi-decision procedure for checking entailment

• has a decidable satisfiability problem for a fragment $\text{Cl}_{\text{dec}}$
Psync: verification correctness

Given a specification $S$ closed under indistinguishability, if a Psync program $P$ satisfies $S$ then the asynchronous semantics of $P$ refines $S$.

$$\text{Clients} \sqsubset \text{Runtime}(P) \sqsubseteq \text{Clients} \sqsubset \text{Spec}(P)$$
### Do Algorithms use Rounds? 

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>LOC</th>
<th>Use rounds</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>One third rule</td>
<td>52</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Last Voting (Paxos)</td>
<td>89</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flood min consensus</td>
<td>24</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Ben-Or randomized consensus</td>
<td>56</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>K-set agreement</td>
<td>42</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>K-set agreement early stopping</td>
<td>33</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Lattice agreement</td>
<td>34</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Two phases commit</td>
<td>53</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Eager reliable broadcast</td>
<td>36</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Chandra Toueg</td>
<td>121</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Generalized Paxos</td>
<td>152</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ViewStampted</td>
<td>91</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Paxos in</td>
<td>LOC</td>
<td>Executable</td>
<td>Verification</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>PSync</td>
<td>89</td>
<td>✓</td>
<td>Semi-</td>
</tr>
<tr>
<td>DistAlgo</td>
<td>43</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Distal</td>
<td>157</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Overlog</td>
<td>107</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>TLA+</td>
<td>53</td>
<td>✗</td>
<td>Interactive</td>
</tr>
<tr>
<td>IO Automata</td>
<td>142</td>
<td>✗</td>
<td>Interactive</td>
</tr>
<tr>
<td>EventML</td>
<td>1729 N</td>
<td>✓</td>
<td>Interactive</td>
</tr>
<tr>
<td>Verdi (Raft)</td>
<td>520</td>
<td>✓</td>
<td>Interactive</td>
</tr>
</tbody>
</table>
### Performance and Verification

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Year</th>
<th>Language</th>
<th>Throughput (x 1000 req./s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Voting in PSync</td>
<td>2015</td>
<td>Scala</td>
<td>170</td>
</tr>
<tr>
<td>Egalitarian Paxos</td>
<td>2013</td>
<td>Go</td>
<td>450</td>
</tr>
<tr>
<td>Paxos in Distal</td>
<td>2013</td>
<td>Scala</td>
<td>150</td>
</tr>
<tr>
<td>JPaxos / SPaxos</td>
<td>2012</td>
<td>Java</td>
<td>75 / 300</td>
</tr>
<tr>
<td>Paxos for system builder</td>
<td>2008</td>
<td>C</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verification of</th>
<th># Invariants (LOC)</th>
<th># VCs</th>
<th>Solving time in s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>One third rule</td>
<td>4 (23)</td>
<td>27</td>
<td>5</td>
</tr>
<tr>
<td>Last Voting</td>
<td>8 (35)</td>
<td>45</td>
<td>16</td>
</tr>
</tbody>
</table>
Conclusions

PSync uses a simple programming abstraction:
Communication-closed rounds
Separates the algorithm from the network
requirements
Asynchrony and faults are modelled by an adversary
that drops messages

Runtime:
Asynchronous semantics refines the lockstep
semantics
Preserves strong consistency
Can be implemented efficiently

Automated verification becomes possible