Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking

E. André, L. Fribourg, R. Soulat and J.-M. Mota

CNRS, THALES, U. Paris 13, ENS Paris-Saclay

January 14, 2019    VMCAI’19    Cascais – Portugal
PLAN

1. Motivation

2. Bully algorithm in the a/synchronous context

3. Adaptation of the Bully algorithm

4. Proofs
   - direct automated proof for a small number $p$ of processes
   - proof with abstractions for $\leq 5000$ processes

5. Conclusion
I. Motivation
Basic facts about leader election algorithms

- Many distributed algorithms need one process to act as a leader or coordinator.
- Does not matter which process does the job, just need to pick one.
- Election algorithm technique to pick a unique coordinator.
- Assumption: each process has a unique ID.
  Goal: find the non-crashed process with the highest ID.
- Problem (Leader election): each node eventually decides whether it is leader or not, subject to the constraint that there is a unique leader.
- Nodes are in one of the three states: leader, follower, candidate.
- When leaving the candidate mode, a node goes into a final state (either leader or follower).
II.

Bully algorithm in the a/synchronous context
Bully algorithm in the a/synchronous setting

- **Topology (here):** complete graph

- **Synchronous case:**
  - All the process clocks are synchronized; processes update their state simultaneously
  - Bully algorithm [Garcia-Molina 1982]: classical synchronous leader election

- **Asynchronous case:**
  - every process is activated *periodically*, but *period* not (exactly) the same for each process (each period takes here its value in [49,51]).
  - besides, the value of each period may slowly evolves (jitter).
  - Initially, the values of clocks are different (setoff).
Short history of asynchronous versions of Bully algorithm

• [GM 1982] claims that the asynchronous version works (with correctness proof similar to the synchronous case).

• [Stoller 1997] gives a counterexample!

• [Svensson 2008] gives a corrected version, but:
  
  – the algorithm requires an important modification
  
  – hundreds of invariants (generated by hand) are needed for the semi-automated proof.
III. A variant of Bully algorithm
General assumptions

• All the IDs of the nodes are different

• Each node has the ability to send messages to all the nodes, and can store messages received from other nodes

• Nodes are either in mode *On* or mode *Off* (failure)

• A node in mode *On* is in one of the states
  • *Follower* (the node is not competing to become leader)
  • *Candidate* (the mode is competing to become leader)
  • *Leader* (the mode has declared itself to be leader)

• Each transmitted message is of the form: *(SenderID, state)* where *state* is the state *On/Off* of the sending node
Updating algorithm (synchronous setting)

At each clock tick, every $On$ process sends to all the other processes its ID number. Each process compares the received ID numbers to its own ID number and updates it.

```plaintext
foreach message ∈ allMessages do
  if message.SenderID > node_i.id then
    state_{next} ← Follower
    higherIDReceived ← true
  if ¬ higherIDReceived then
    if node_i.state = Follower then
      state_{next} ← Candidate
    else if node_i.state = Candidate then
      state_{next} ← Leader
    else if node_i.state = Leader then
      state_{next} ← Leader
  node_i.state ← state_{next}
```

Property P to be proven:
After a certain number of clean rounds (rounds with no crash and no recovery),

- the process $On$ with the higher ID is Leader, and
- all the other $On$ processes are Follower (no $On$ process is Candidate)
Complications (asynchronous setting)

• If clock ticks are not synchronized, the messages are not emitted (and received) simultaneously
Complications due to asynchronous clocks

Table 2: Jitter values for Example 1

<table>
<thead>
<tr>
<th></th>
<th>jitter¹</th>
<th>jitter²</th>
<th>jitter³</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1</td>
<td>0.5</td>
<td>-0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Node 2</td>
<td>0</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>Node 3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Fig. 1: Activation of three nodes with uncertain periods and jitters

• nb of activations for nodes 1 and 3 always the same up to a difference of 1 (due to the jitters) because they have same periods.

• But nb of activations for node 2 becomes smaller than that of nodes 1 and 3 by an increasing difference, since node 2 is slower (period: 51 instead of 49).

• This phenomenon does not occur when periods are equal for all nodes, and makes this setting more challenging.
A simple solution

- To overcome this difficulty, each ID proceeds to the update not at each period end, but every two (or more) periods.

- Basic insight:

Lemma 1. Assume a node $i$ and activation times $t_i^j$ and $t_i^{j+2}$. Then in between these two activations, node $i$ received at least one message from all nodes.
Basic assumptions

- **Instantiated model with uncertainty**
  - *Periods* and *jitters* are known to belong to given intervals.

<table>
<thead>
<tr>
<th>Table 1: Constants (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constant</strong></td>
</tr>
<tr>
<td>period(_{\text{min}})</td>
</tr>
<tr>
<td>period(_{\text{max}})</td>
</tr>
<tr>
<td>jitter(_{\text{min}})</td>
</tr>
<tr>
<td>jitter(_{\text{max}})</td>
</tr>
</tbody>
</table>

- the number \( p \) of processes is **given**
- The algorithm should work for \( p \) as **large** as possible
Extended Bully algorithm

```
Algorithm 1: UpdateNode(i)

1 if node_i.EvenActivation then
2     allMessages ← ReadMailbox()
3     higherIDReceived ← false
4     foreach message ∈ allMessages do
5         if message.SenderID > node_i.id then
6             state_next ← Follower
7             higherIDReceived ← true
8     if not higherIDReceived then
9         if node_i.state = Follower then
10            state_next ← Candidate
11         else if node_i.state = Candidate then
12            state_next ← Leader
13         else if node_i.state = Leader then
14            state_next ← Leader
15         node_i.state ← state_next
16 node_i.EvenActivation ← ¬node_i.EvenActivation
17 message = {node_i.id; node_i.state}
18 Send_To_All_Network(message)
```
Objective

• **Definition 1** (round). A *round* is a time period during which all the nodes that are *On* have sent at least one message.

• **Definition 2** (cleanness). A round is said to be *clean* if during its time period no node have been switched from *On* to *Off* or from *Off* to *On*.

The correctness property *P* that we want to prove automatically is:

« After 4 clean nodes, the node with the highest ID is recognized as the leader by all the *On* nodes of the network. »
IV. PROOFS
IV.1 Direct proof of $P$ using SMT solving

- Using a model $M$ of the algorithm, we get automatically a proof of $P$ using SMT solver SafeProver [EJ17] when $p$ is small ($p \leq 5$).

- This leads us to consider a method using abstractions to prove $P$ for large values of $p$. 
IV.2 Proof with abstractions

we consider two abstractions of $M$

- 1st abstraction $M^*$ consists in considering one of the $p$ processes (arbitrarily), and consider the set of other processes under the form of a single big automaton (no timing information)

- In the 2nd abstraction $T$, one considers two generic processes under the form of timed automaton with one parameter (the fixed value of the period lying in $[49,51]$)

we also decompose property $P$ into several properties $P_1$-$P_2$-$P_3$-$P_4$. 
Scheme of the proof

For a given number $p$ of processes, prove:

- $P1$-$P2$ on $M^*$ with SMT solver (SafeProver)

- $P3$ on $T$ with parametric timed model checker (IMITATOR)
  [NB: exact statement of $P3$ depends on values of periods and jitters]

- $P4$ on $M^*$ with SMT solver using $P1$-$P2$-$P3$ as lemmas

Method works for $p = 5000$!
Automated proof of $P1-P2$ for $M^*$ using SMT solver SafeProver

Scheme of model $M^*$ with node $i$ under study interacting with other nodes

- $P1$: $(\text{Activation}(j) \geq 2 \land node_j.id \neq \text{maxId}) \Rightarrow node_j.state = \text{Follower}$
- $P2$: $(\text{Activation}(j) \geq 2 \land node_j.id = \text{maxId})$
  \[ \Rightarrow node_j.state \in \{\text{Candidate, Leader}\} \]
Automated proof of \( P3 \) for \( T \) using parametric timed model checker IMITATOR

\[
\begin{align*}
Activation(i) & := 0 \\
0 & \leq c_i \leq per_i + jitter_{\text{max}} \\
\text{node}_i & \\
c_i & \geq per_i + jitter_{\text{min}} \\
Activation(i) & := Activation(i) + 1
\end{align*}
\]

Fig. 3: Component 1 of timed model \( T \)

For nodes \( \text{node}_i \) and \( \text{node}_j \), the property that we want to specify corresponds in the direct model \( M \) (without abstraction) of Section 3 to:

\[- (\text{Activation}(i) \leq 13 \; \land \; \text{Activation}(j) \leq 13) \implies |\text{Activation}(i) - \text{Activation}(j)| \leq 2.\]

In our timed abstract model \( T \), such a property becomes:

\[- (P3): \forall i \in \{1, \ldots, p\} \; \text{Activation}(j) \leq 13 \implies -2 \leq \text{Activation}(j) - \text{Activation}(i) \leq 1.\]

where \( \text{Activation}(i) \) denotes the number of activations of node \( i \) since the last clean round.
Automated proof of $P4$ for $M^*$ using SMT solver with $P1$-$P2$-$P3$ as assumptions
Conclusion and final remarks

• We considered an asynchronous leader election algorithm

• We proved automatically its correctness property $P$ using SMT solving for a small number $p$ of nodes

• Using two abstractions and a decomposition of $P$, we verify the algorithm using SMT and parametric timed model checking for $p$ up to 5000.

• The algorithm considered here is actually a variant of the original algorithm designed by THALES (not available for confidentiality reasons).

• The same kind of proof has been done for the original algorithm

• We are now considering to prove formally the correctness of the two abstractions
THANKS!