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 Cascai – 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 <= 5000 processes

5. Conclusion

I. Motivation

Basic facts about leader election algorithms

- Many distributed algorithms needs 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 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 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

```
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

	$jitter^1$	$jitter^2$	$jitter^3$
Node 1	0.5	−0.5	0.5
Node 2	0	0.1	0
Node 3	0.1	0.3	0.5

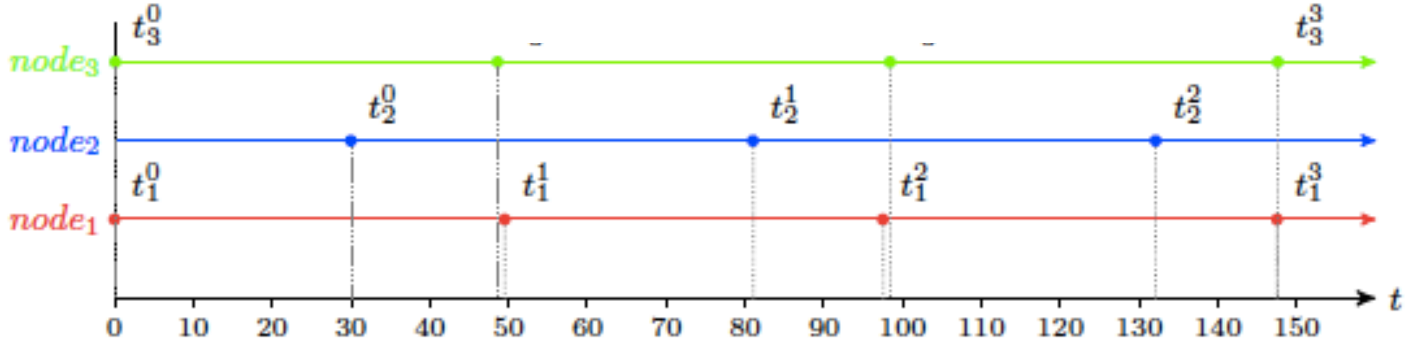


Fig. 1: Activation of three nodes with uncertain periods and jitters

- 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 1: Constants (in ms)

Constant	Value
period_{min}	49
period_{max}	51
jitter_{min}	-0.5
jitter_{max}	0.5

  - the number p of processes is given
  - The algorithm should work for p as large as possible

Extended Bully algorithm

Algorithm 1: *UpdateNode(i)*

```
if node_i.EvenActivation then
    allMessages ← ReadMailbox()
    higherIDreceived ← false
    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
node_i.EvenActivation ← ¬node_i.EvenActivation
message = {node_i.id; node_i.state}
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 ($p <= 5$).

- 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*
  - 1^{st} 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 2^{nd} 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 *P1-P2-P3-P4* .

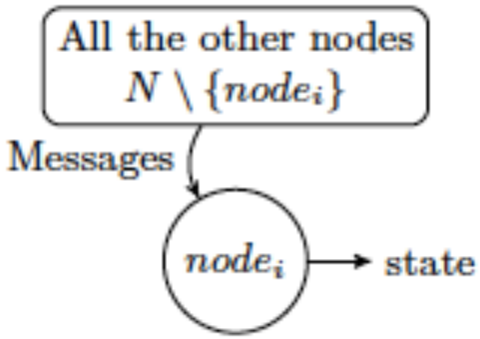
Scheme of the proof

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: $(Activation(j) \geq 2 \ \wedge \ node_j.id \neq maxId) \Rightarrow \ node_j.state = \text{Follower}$
- P2: $(Activation(j) \geq 2 \ \wedge \ node_j.id = maxId)$
  $\Rightarrow node_j.state \in \{\text{Candidate, Leader}\}$

Automated proof of *P3* for *T* using parametric timed model checker IMITATOR

Fig. 3: Component 1 of timed model T

For nodes $node_i$ and $node_j$, the property that we want to specify corresponds in the direct model M (without abstraction) of Section 3 to:

- $(Activation(i) \leq 13 \ \wedge \ Activation(j) \leq 13)$
$$\Rightarrow |\ Activation(i) - Activation(j)\ | \leq \ 2.$$

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

- (P3): $\forall i \in \{1, \ldots, p\}$ $Activation(j) \leq 13 \Rightarrow$
$-2 \leq Activation(j) - Activation(i) \leq 1.$

where $Activation(i)$ 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

$$P4 : (Activation(i) \geq 4 \ \wedge \ node_i.id = maxId) \Rightarrow node_i.state = \text{Leader}$$

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!