# Formal Verification of an Industrial Distributed Algorithm: an Experience Report

Nikolai Kosmatov, Delphine Longuet, Romain Soulat

Thales Research & Technology, Palaiseau, France

(thanks to Etienne André, Laurent Fribourg, and Jean-Marc Mota
for their contribution to a previous case study)

ISOLA 2021, Rhodes, October 25, 2021

# Context: Verification of Distributed Algorithms

- Distributed algorithms include consensus protocols
  - the processes (nodes) of a network, executing the same code, have to come to the agreement on some data
  - Ex: Leader election, identification of working nodes
- An active research area: several verified algorithms exist
  - Synchronous: Bully algorithm [Garcia-Molina, 1982]
  - Asynchronous: Lamport’s leader election protocol [1998]
    - proved e.g. in TLA+, UPPAAL
- As each protocol is tightly linked to the considered setting, engineers often need to verify other protocols
- Thales designed several (confidential) consensus protocols whose properties had to be verified

# Target Protocol Characteristics

- The system is composed of p identical computing nodes
  - Nodes can perform various tasks and receive a part of the workload
- The nodes are fully interconnected
  - any node can send messages to any other node to communicate computation results
- Periodically, each node sends to all other nodes a special state message indicating that the sender is still alive and providing some additional data
- The algorithm uses these messages to compute a list of all working nodes in the network
  - Used for workload balancing, clock synchronization, leader election, etc.
- Local uncertainty and time variations modeled
  - Each node’s period is within predefined bounds, and activation time can be perturbed by jitters
- A node sends a state message every second activation

# Example 1: Periods, jitters and activation time

The $j$-th activation of node $node_i$ occurs at time $t_i^j = t_i^{j-1} + node_i.per + jitter_i^j$ for $j > 0$. We set besides: $t_i^0 = node_i.start$.

| Constant | Value |
|---|---|
| $\mathsf{period_{min}}$ | 49 |
| $\mathsf{period_{max}}$ | 51 |
| $\mathsf{jitter_{min}}$ | −0.5 |
| $\mathsf{jitter_{max}}$ | 0.5 |
| $\mathsf{msgDelay_{min}}$ | 0 |
| $\mathsf{msgDelay_{max}}$ | 0 |

| Node | $per$ | $start$ | $jitter_i^1$ | $jitter_i^2$ | $jitter_i^3$ |
|---|---|---|---|---|---|
| $node_1$ | 49 | 0 | 0.5 | −0.5 | 0.2 |
| $node_2$ | 51 | 30 | 0 | 0.1 | 0 |
| $node_3$ | 49 | 0.1 | 0.1 | −0.5 | 0.5 |

**Fig. 1.** (a) Static constants (in ms), and (b) values chosen for the nodes in Example 1.

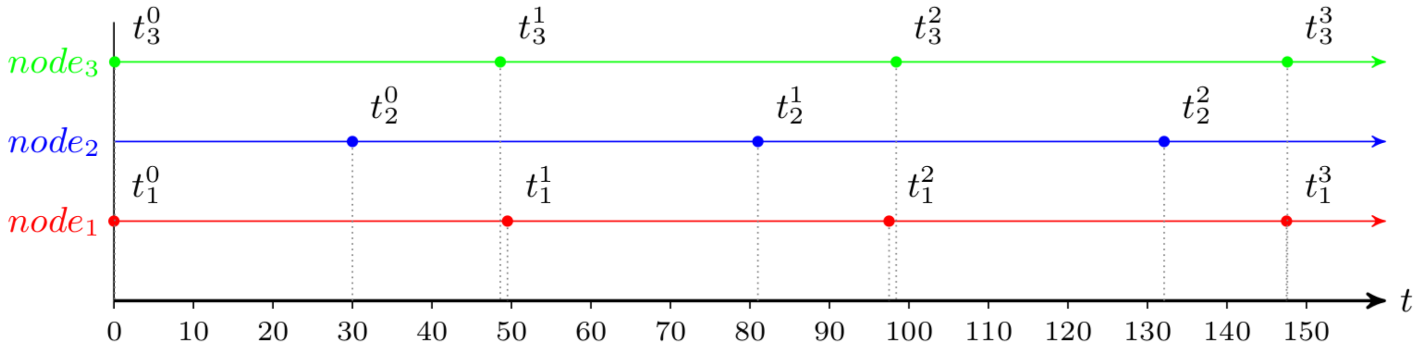


**Fig. 2.** Activation times (in ms) of the three nodes of Example 1.

# Target Properties

- Final property ($P_\alpha$): all working nodes reach a consensus about the set of working nodes after α rounds (in our algorithm, α = 7)

$$\forall k \in \{1,\ldots,p\},\ (\ node_k \in networkState\ \wedge Activation_k \geq \alpha\ ) \Rightarrow node_k.networkView = networkState \qquad (P_\alpha)$$

- Intermediate properties ($P_j$): Some partial knowledge ("likely information") after j rounds

$$\forall k \in \{1,\ldots,p\},\ (\ node_k \in networkState\ \wedge Activation_k \geq j\ ) \Rightarrow \ldots \qquad (P_j)$$

- In our algorithm, only P2, P5, P7 are stronger than the previous ones

# Verification Methodology: Overview

- Use a model $M_{sim}$ simulating all nodes with all possible interleavings
  - quickly check expected properties
  - detect counter-examples
  - prove for a small number of nodes (up to 4)
  - does not scale for bigger number of nodes (>20)
- Use an abstact model $M_{abs}$ to simulate a unique node, assume properties about other nodes and timing related properties
  - prove for bigger number of nodes
  - detect counter-examples
- Use a specific timing model $M_T$ to establish timing related properties

# Model $M_{sim}$

- Simulates the whole network
- Explicitly represents all nodes
- Activates nodes in a loop

```
// Network initialization for nodes i=1,2,...,p
State : node_i.state
Integer : node_i.id, node_i.per, node_i.start, Activation_i, nextActivationTime_i
Boolean :
  node_i.EvenActivation, node_i.failure, node_i.rcvFailure, node_i.sndFailure
Assume : AllDifferent(node_i.id | i ∈ 1, ..., p)  // Identifiers are unique
foreach i ∈ {1, ..., p} do
    Activation_i ← 0
    Assume : period_min ≤ node_i.per ≤ period_max
    Assume : 0 ≤ node_i.start < node_i.per
    nextActivationTime_i ← node_i.start
// Mailbox initialization
...
// Main algorithm simulating the network execution
while true do
    i ← indexMin(nextActivationTime)  // Node with the smallest time
    if ¬node_i.failure then
        UpdateNode(i)  // Execute the node
        Activation_i ← Activation_i + 1
    jitter ← nondet()  // Choose an arbitrary value for a new jitter
    Assume : jitter_min ≤ jitter ≤ jitter_max  // in the considered bounds
    nextActivationTime_i ← nextActivationTime_i + node_i.per + jitter
    Assert : P_1 ∧ ··· ∧ P_α  // Partial and final properties
```

# Model $M_{abs}$

- Models one node
- Assumptions on other nodes
- Assumptions for timing properties
- While proving $P_l$ for $node_i$ , we assume $P_1,\ldots,P_{l-1}$ for other nodes

```
// Initialization for nodes k=1,2,...,p
Integer : node_k.id, Activation_k
Boolean : node_k.failure, node_k.rcvFailure, node_k.sndFailure
Assume : AllDifferent(node_k.id | k ∈ 1,...,p)  // Identifiers are unique
// Initialization for node i
Assume : i ∈ {1,...,p}
Activation_i ← 0
Boolean : node_i.EvenActivation
// Main loop iteratively activating node i
while true do
    Mailbox ← ∅  // Model possible messages from other nodes
    for k ∈ {1,...,p} \ {i} do
        message_k ← nondet()
        if ¬node_i.rcvFailure ∧ ¬node_k.sndFailure then
            Mailbox ← Mailbox ∪ message_k
        Activation_k ← Activation_k + |nondet()| // An increasing value
    Assume : P_timed ∧ P_1 ∧ ··· ∧ P_{l−1} // Assume up to P_{l−1} for all nodes
    if ¬node_i.failure then
        UpdateNode(i)
    Activation_i ← Activation_i + 1
    Assert : (P_1 ∧ ··· ∧ P_l)|_{node_i}  // Prove properties up to P_l for node_i
```

# Timing properties (Model $M_T$)

- Property $P_{timed}$ relates the number of executions of two nodes for given system parameters
- Proved using a parametric timed automaton, an extension of a timed automaton, in IMITATOR model-checker

$$\begin{aligned}&\forall i,k \in \{1,\ldots,p\},\ Activation_i \leq \beta \\ &\quad \Rightarrow \ |\ Activation_i - Activation_k\ |\ \leq\ \gamma\end{aligned} \qquad (P_{timed})$$

# Imprecision due to Abstraction

- In model $M_{sim}$ the consensus was reached after $\alpha = 7$ activations
- In the abstract model $M_{abs}$ it was reached after $\alpha = 8$ activations
- This extra delay is due to abstraction
- It was not an issue for system developers, a rigorous proof being more important.
- Therefore, we use $\alpha = 8$ in the specification and verification of $M_{abs}$
- The other key properties ($P_2$ and $P_5$) were true for the same j

# Experiments

- Three tools: SafeProver (by SafeRiver), CBMC, KLEE
- Run on both models $M_{sim}$ and $M_{abs}$
- Goal: record the results that industrial engineers can obtain
  - without an advanced knowledge of these tools
  - on a real-life distributed algorithm
- The goal was NOT to compare the tools or to judge their potential

# Results with SafeProver

(a)

| #nodes<br>variant | $p = 3$<br>Correct | $p = 4$<br>Correct | $p = 5$<br>Correct |
|---|---|---|---|
| time | 59.5 s | 95m48 s | TO |
| result | ✓ | ✓ | — |

(b)

| #nodes<br>variant | $p = 3$<br>Correct | $p = 18$<br>Correct | $p = 42$<br>Correct | $p = 100$<br>Correct |
|---|---|---|---|---|
| time | 0.29 s | 9.74 s | 5min12 s | 15min30 |
| result | ✓ | ✓ | ✓ | ✓ |

**Fig. 3.** Experiments with SafeProver for correct properties with all failure modes for models (a) $M_{\text{sim}}$, and (b) $M_{\text{abs}}$. TO means a timeout (set to 2 hours).

# Results with CBMC without failures

| #nodes | $p = 3$ | | | | $p = 4$ | | | | $p = 5$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| variant | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | Correct | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | Correct | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | Correct |
| time | 0.25 s | 0.95 s | 6.27 s | 37.75 s | 0.33 s | 1.52 s | 53.98 s | 14 m52 s | 0.57 s | 8.4 s | 2 m27 s | TO |
| result | CE | CE | CE | ✓ | CE | CE | CE | ✓ | CE | CE | CE | — |
| RDP | 0.13 s | 0.74 s | 5.76 s | 37.49 s | 0.20 s | 1.25 s | 53.18 s | 14 m51 s | 0.35 s | 7.95 s | 2 m26 s | TO |
| #vars | 30,898 | 70,488 | 96,542 | 87,797 | 47,765 | 111,735 | 153,802 | 143,204 | 68,448 | 162,716 | 224,677 | 212,182 |
| #clauses | 110,966 | 256,340 | 351,902 | 319,867 | 172,625 | 408,119 | 562,800 | 523,886 | 261,475 | 627,154 | 867,407 | 819,048 |

**Fig. 4.** Experiments on erroneous and correct versions of model $M_{sim}$ simulating all nodes without failures with CBMC. TO means a timeout (set to 2 hours). RDP stands for runtime decision procedure.

| #nodes | $p = 3$ | | | | $p = 18$ | | | | $p = 42$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| variant | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | Correct | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | Correct | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | Correct |
| time | 0.32 s | 0.33 s | 0.41 s | 0.34 s | 2.19 s | 2.35 s | 2.42 s | 2.85 s | 8.07 s | 8.65 s | 9.81 s | 11.05 s |
| result | CE | CE | CE | ✓ | CE | CE | CE | ✓ | CE | CE | CE | ✓ |
| RDP | 0.13 s | 0.14 s | 0.17 s | 0.14 s | 0.88 s | 0.97 s | 1.08 s | 1.18 s | 2.20 s | 2.79 s | 3.81 s | 4.28 s |
| #vars | 38,615 | 38,606 | 38,597 | 38,594 | 197,795 | 197,786 | 197,777 | 197,774 | 452,483 | 452,474 | 452,465 | 452,462 |
| #clauses | 116,705 | 116,411 | 116,009 | 115,851 | 578,390 | 577,736 | 576,434 | 575,856 | 1,317,086 | 1,315,856 | 1,313,114 | 1,311,864 |

**Fig. 5.** Experiments on erroneous and correct versions of the abstract model $M_{abs}$ without failures with CBMC. TO means a timeout (set to 2 hours).

# Results with CBMC with failures

(a)

| #nodes variant | $p = 3$ Correct | $p = 4$ Correct |
|---|---|---|
| time | 4 min20 s | TO |
| result | ✓ | — |
| RDP | 4 min19 s | TO |
| #vars | 305,431 | 527,197 |
| #clauses | 986,574 | 1,713,053 |

(b)

| #nodes variant | $p = 3$ Correct | $p = 18$ Correct | $p = 22$ Correct | $p = 23$ Correct |
|---|---|---|---|---|
| time | 2.36 s | 9 min2 s | 55 min47 s | TO |
| result | ✓ | ✓ | ✓ | — |
| RDP | 1.63 s | 8 min46 s | 55 min19 s | TO |
| #vars | 371,265 | 2,007,225 | 2,436,945 | 2,543,945 |
| #clauses | 1,143,416 | 6,080,111 | 7,350,811 | 7,665,476 |

**Fig. 6.** Experiments with CBMC on correct versions of models (a) $M_{\text{sim}}$ with failures, and (b) $M_{\text{abs}}$ with failures. TO means a timeout (set to 2 hours).

# Results with KLEE

| #nodes | $p=2$ | | | $p=3$ | | | $p=4$ | | | $p=5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| variant | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_6^{err}$ |
| time | 0.6s | 7s | 48s | 0.8s | 12m33s | TO | 1.15s | TO | TO | 2.1s | TO | TO |
| result | CE | CE | CE | CE | CE | — | CE | — | — | CE | — | — |
| #instr. | 2,299 | 595,279 | 4,231,836 | 4,820 | 51,662,006 | ? | 12,288 | ? | ? | 44,118 | ? | ? |

**Fig. 7.** Experiments on erroneous versions of model $M_{sim}$ simulating all nodes with KLEE. For correct versions, the tool timed out. TO means a timeout (set to 2 hours).

| #nodes | $p=2$ | | | $p=3$ | | | $p=4$ | | | $p=5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| variant | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ | $P_1^{err}$ | $P_4^{err}$ | $P_7^{err}$ |
| time | 0.3s | 27s | 17m53s | 0.5s | 26m21s | TO | 1s | 60m53s | TO | 1.7s | 1h25m | TO |
| result | CE | CE | CE | CE | CE | — | CE | CE | — | CE | CE | — |
| #instr. | 2,213 | 2,039,992 | 57,289,534 | 6,739 | 63,235,648 | ? | 21,582 | 88,588,978 | ? | 62,962 | 109,841,990 | ? |

**Fig. 8.** Experiments on erroneous and correct versions of the abstract model $M_{abs}$ with KLEE. For correct versions, the tool timed out. TO means a timeout (set to 2 hours).

# Lessons Learned (1/5)

- Several consensus algorithms were verified
- Industrial users often need to verify a specific algorithm
  - Prefer to prove the existing (or slightly adapted) legacy algorithm
  - Existing algorithms may not meet target system constraints
    - memory size, network usage, computational time, non-interference with other computations, relevant fault models and robustness constraints, the level of possible variations of the activation or communication times.

# Lessons Learned (2/5)

- After verifying one algorithm, the engineer often needs to adapt it to a new system and to verify again
- Generic verification methodologies applicable to large families of similar algorithms are required
- We present such a methodology for a family of consensus algorithms
- Criteria for acceptance of the methodology include
  - Capacity to perform the proof
  - Possibility to analyze the real-life code or have a model very similar to the code
  - Possibility to produce and easily read counter-examples

# Lessons Learned (3/5)

- Models of consensus algorithms have a high combinatorial complexity
  - due to several free variables in the initial state, lots of possible interleavings...
- Complexity highly increases with the number of nodes and executions
- Symbolic tools seem to be most suitable for such algorithms
  - symbolic model checking and symbolic execution
- Timed model checking alone was not sufficient in our experiments
  - Timed model checkers we tried did not scale
  - Modeling language must be close to the code
  - Need to easily generate and read counter-examples

# Lessons Learned (4/5)

- Symbolic model checkers (SafeProver and CBMC) very powerful both for finding counter-examples and proving the correct version of the algorithm
  - Support of bit operations was particularly useful
  - A compact bit-level encoding of data improved the results
- Symbolic execution with Klee
  - Very useful to detect counter-examples for small numbers of nodes / executions
  - Due to combinatorial explosion, cannot explore all paths to show the absence of errors on the correct models

# Lessons Learned (5/5)

- Abstracting the system model using abstraction was essential to scale for a large number of nodes
  - Proof on the complete model $M_{sim}$ worked for few nodes ($p < 5$),
  - But it ran out of time and memory for bigger numbers of nodes required in the target systems
- The rely-guarantee based approach (dating back to [Jones,1983]) solved this issue for the algorithms we faced

# Future Work

- Application of the methodology to other industrial algorithms
- Proof of the assumptions for the real-life C code using deductive verification (e.g. in Frama-C)
- Experiences using other verification tools (model checking and symbolic execution)

More generally,

- Collecting the engineers’ needs and applying recent software verification advancements to industrial projects remains a priority for the formal methods group of Thales Research and Technology

# Back-Up Slides

# State Update & Message Computation

**Algorithm 1:** Pseudo-code of function $UpdateNode(i)$

**if** $node_i.EvenActivation$ **then**

  $allMessages \leftarrow ReadMessages(i)$

  $nodeState \leftarrow ComputeState(allMessages, nodeState)$

  $message \leftarrow ComputeMessage(nodeState)$

**if** $\neg node_i.sndFailure$ **then**

  $SendToAllNetwork(message, currentTime)$

$node_i.EvenActivation \leftarrow \neg node_i.EvenActivation$

# Fault Models

- F1: A node can stop and flush its internal memory. When restarting, the node will believe it is alone in the network until it receives messages from the other nodes. This corresponds to a node shutting down.
- F2: A node can stop and keep its internal memory. When restarting, the node will have the same state as before stopping, it will still assume the network in the same state as when it stopped, until it receives messages that will contradict this belief. This corresponds to a node freezing.
- F3: A node can stop sending and receiving messages. This corresponds to a disconnection from the network.
- F4/F5: A node can stop receiving (resp., sending) messages but still be able to emit (resp. receive) messages. This corresponds to a partial disconnection.