Parameterized Model Checking of Fault-tolerant Distributed Algorithms

Annu Gmeiner  Igor Konnov  Ulrich Schmid  Helmut Veith
Josef Widder

PV 2014
Rome, Sept 6, 2014
Why fault-tolerant (FT) distributed algorithms

defaults not in the control of system designer

- bit-flips in memory
- power outage
- disconnection from the network
- intruders take control over some computers
Why fault-tolerant (FT) distributed algorithms

faults not in the control of system designer

- bit-flips in memory
- power outage
- disconnection from the network
- intruders take control over some computers

distributed algorithms intended to make systems more reliable even in the presence of faults

- replicate processes
- exchange messages
- do coordinated computation
- goal: keep replicated processes in “good state”
Fault-tolerant distributed algorithms

- $n$ processes communicate by messages

$n > 3t \land t \geq f \geq 0$
$n$ processes communicate by messages

all processes know that at most $t$ of them might be faulty
Fault-tolerant distributed algorithms

- $n$ processes communicate by messages
- all processes know that at most $t$ of them might be faulty
- $f$ are actually faulty
- resilience conditions, e.g., $n > 3t \land t \geq f \geq 0$
Fault-tolerant DAs: Model Checking Challenges

- unbounded data types
  counting how many messages have been received

- parameterization in multiple parameters
  among \( n \) processes \( f \leq t \) are faulty with \( n > 3t \)

- contrast to concurrent programs
  fault tolerance against adverse environments

- degrees of concurrency
  many degrees of partial synchrony

- continuous time
  fault-tolerant clock synchronization
Importance of liveness in distributed algorithms

Interplay of safety and liveness is a central challenge in DAs

- Interplay of safety and liveness is non-trivial
- Asynchrony and faults lead to impossibility results
Importance of liveness in distributed algorithms

Interplay of safety and liveness is a central challenge in DAs

- interplay of safety and liveness is non-trivial
- asynchrony and faults lead to impossibility results

Rich literature to verify safety (e.g. in concurrent systems)

Distributed algorithms perspective:

- “doing nothing is always safe”
- “tools verify algorithms that actually might do nothing”
Model checking problem for fault-tolerant DA algorithms

- given a distributed algorithm and spec. $\varphi$
- system description:
  
  $M(n, t, f) = P(n, t, f) \parallel P(n, t, f) \parallel \cdots \parallel P(n, t, f)$

- every $M(n, t, f)$ is a system of $n - f$ correct processes
- show for all $n, t, f$ satisfying $n > 3t \land t \geq f \geq 0$

  $M(n, t, f) \models \varphi$

---

Josef Widder (www.forsyte.at)

Parameterized Model Checking of FTDAs

PV 2014
Model checking problem for fault-tolerant DA algorithms

- given a distributed algorithm and spec. $\varphi$
- system description:
  \[ M(n, t, f) = P(n, t, f) \parallel P(n, t, f) \parallel \cdots \parallel P(n, t, f) \]
- every $M(n, t, f)$ is a system of $N(n, t, f)$ correct processes
- show for all $n$, $t$, and $f$ satisfying resilience condition $M(n, t, f) \models \varphi$
Properties in Linear Temporal Logic

Unforgeability (U). If \( v_i = 0 \) for all correct processes \( i \), then for all correct processes \( j \), \( \text{accept}_j \) remains 0 forever.

\[
G \left( \left( \bigwedge_{i=1}^{n-f} v_i = 0 \right) \rightarrow G \left( \bigwedge_{j=1}^{n-f} \text{accept}_j = 0 \right) \right)
\]

Completeness (C). If \( v_i = 1 \) for all correct processes \( i \), then there is a correct process \( j \) that eventually sets \( \text{accept}_j \) to 1.

\[
G \left( \left( \bigwedge_{i=1}^{n-f} v_i = 1 \right) \rightarrow F \left( \bigvee_{j=1}^{n-f} \text{accept}_j = 1 \right) \right)
\]

Relay (R). If a correct process \( i \) sets \( \text{accept}_i \) to 1, then eventually all correct processes \( j \) set \( \text{accept}_j \) to 1.

\[
G \left( \left( \bigvee_{i=1}^{n-f} \text{accept}_i = 1 \right) \rightarrow F \left( \bigwedge_{j=1}^{n-f} \text{accept}_j = 1 \right) \right)
\]
Properties in Linear Temporal Logic

**Unforgeability (U).** If $v_i = 0$ for all correct processes $i$, then for all correct processes $j$, $\text{accept}_j$ remains 0 forever.

$$G \left( \left( \bigwedge_{i=1}^{n-f} v_i = 0 \right) \rightarrow G \left( \bigwedge_{j=1}^{n-f} \text{accept}_j = 0 \right) \right)$$

**Safety**

**Completeness (C).** If $v_i = 1$ for all correct processes $i$, then there is a correct process $j$ that eventually sets $\text{accept}_j$ to 1.

$$G \left( \left( \bigwedge_{i=1}^{n-f} v_i = 1 \right) \rightarrow F \left( \bigvee_{j=1}^{n-f} \text{accept}_j = 1 \right) \right)$$

**Liveness**

**Relay (R).** If a correct process $i$ sets $\text{accept}_i$ to 1, then eventually all correct processes $j$ set $\text{accept}_j$ to 1.

$$G \left( \left( \bigvee_{i=1}^{n-f} \text{accept}_i = 1 \right) \rightarrow F \left( \bigwedge_{j=1}^{n-f} \text{accept}_j = 1 \right) \right)$$

**Liveness**
Threshold-guarded fault-tolerant distributed algorithms
Threshold-guarded FTDAs

Fault-free construct: quantified guards \((t=f=0)\)

- Existential Guard
  
  if received \(m\) from \(some\) process then \(...\)

- Universal Guard
  
  if received \(m\) from \(all\) processes then \(...\)

These guards allow one to treat the processes in a parameterized way.
Threshold-guarded FTDAs

Fault-free construct: quantified guards \((t=f=0)\)

- Existential Guard
  - if received \(m\) from \(\text{some}\) process then ...
- Universal Guard
  - if received \(m\) from \(\text{all}\) processes then ...

These guards allow one to treat the processes in a parameterized way

*what if faults might occur?*
Fault-free construct: quantified guards ($t=f=0$)

- **Existential Guard**
  if received $m$ from *some* process then ...

- **Universal Guard**
  if received $m$ from *all* processes then ...

These guards allow one to treat the processes in a parameterized way.

*what if faults might occur?*

**Fault-Tolerant Algorithms:** $n$ processes, at most $t$ are Byzantine

- **Threshold Guard**
  if received $m$ from $n-t$ processes then ...

- *(the processes cannot refer to $f$!)*
Counting argument in threshold-guarded algorithms

Correct processes count distinct incoming messages

if received $m$ from $t+1$ processes then ...

$n$
$t$
$t+1$
Counting argument in threshold-guarded algorithms

Correct processes count distinct incoming messages

if received $m$ from $t+1$ processes then ...
Counting argument in threshold-guarded algorithms

Correct processes count distinct incoming messages

If received \( m \) from \( t+1 \) processes then ...

At least one non-faulty sent the message.
our abstractions
at a glance
Data + counter abstraction over parametric intervals

\[ n = 6, \ t = 1, \ f = 1 \]

\[ t + 1 = 2, \ n - t = 5 \]

nr. processes (counters)

1 process at (accepted, received=5)

3 processes at (sent, received=3)
Data + counter abstraction over parametric intervals

\[ n = 6, \ t = 1, \ f = 1 \]
\[ t + 1 = 2, \ n - t = 5 \]

nr. processes (counters)
$n = 6, \ t = 1, \ f = 1$

$t + 1 = 2, \ n - t = 5$

nr. processes (counters)
Data + counter abstraction over parametric intervals

\[ n = 6, \ t = 1, \ f = 1 \]

\[ n > 3 \cdot t \land t \geq f \]

nr. processes (counters)

Parametric intervals:

\[ I_0 = [0, 1) \quad I_1 = [1, t + 1) \]

\[ I_{t+1} = [t + 1, n - t) \]

\[ I_{n-t} = [n - t, \infty) \]
Data + counter abstraction over parametric intervals

\[ n > 3 \cdot t \land t \geq f \]

nr. processes (counters)

Parametric intervals:

\[ I_0 = [0, 1) \quad I_1 = [1, t + 1) \]

\[ I_{t+1} = [t + 1, n - t) \]

\[ I_{n-t} = [n - t, \infty) \]

all correct processes accepted?
Related work: \((0, 1, \infty)\)-counter abstraction

Pnueli, Xu, and Zuck (2001) introduced \((0, 1, \infty)\)-counter abstraction:

- finitely many local states,
  
ed.g., \(\{N, T, C\}\).

- **abstract** the number of processes in every state,
  
ed.g., \(K : C \mapsto 0, \quad T \mapsto 1, \quad N \mapsto \text{"many"}\).

- perfectly reflects mutual exclusion properties
  
ed.g., \(G (K(C) \neq \text{"many"})\).
Related work: \((0, 1, \infty)\)-counter abstraction

Pnueli, Xu, and Zuck (2001) introduced \((0, 1, \infty)\)-counter abstraction:

- finitely many local states, e.g., \(\{N, T, C\}\).
- abstract the number of processes in every state, e.g., \(K: C \mapsto 0, \ T \mapsto 1, \ N \mapsto \text{“many”}\).
- perfectly reflects mutual exclusion properties e.g., \(G (K(C) \neq \text{“many”})\).

Our parametric data + counter abstraction:

- unboundedly many local states (nr. of received messages)
- finer counting of processes:
  \(t + 1\) processes in a specific state can force global progress, while \(t\) processes cannot

- mapping \(t, t + 1, \) and \(n - t\) to “many” is too coarse.
Tool Chain: ByMC

Parametric Promela code → STATIC ANALYSIS + YICES

Parametric Interval Domain $\hat{D}$

Parametric Promela code

PARAMETRIC DATA ABSTRACTION with YICES

Parametric Promela code

PARAMETRIC COUNTER ABSTRACTION with YICES

normal Promela code

SPIN

property holds

counterexample

Figure: The abstraction scheme
Tool Chain: **BYMC**

**Parametric Promela code** → **Static analysis + Yices**

**Parametric Interval Domain \( \hat{D} \)**

**Parametric data abstraction with Yices**

**Parametric Promela code** → **Concrete counter representation (VASS)**

**SMT formula**

**Parametric counter abstraction with Yices**

**Normal Promela code** → **Refine**

**Yices**

**Spin**

- Property holds
- Counterexample

- Unsat
- Sat

- Counterexample feasible
Tool Chain: ByMC

Parametric Promela code → STATIC ANALYSIS + YICES

Parametric Interval Domain $\hat{D}$

PARAMETRIC DATA ABstraction WITH YICES

Parametric Promela code

PARAMETRIC COUNTER ABstraction WITH YICES

Concrete counter representation (VASS)

SMT formula

Invariant candidates (by the user)

Refine

YICES

sat

unsat

normal Promela code

SPIN

property holds

counterexample

counterexample feasible
Parameterized model checking performs well (the red line).
Experiments for fixed parameters quickly degrade ($n = 9$ runs out of memory).
We found counter-examples for the cases $n = 3t$ and $f > t$, where the resilience condition is violated.
Completeness threshold for bounded model checking

Fix a threshold automaton TA and a size function $N$.

**Theorem**

For each $p$ with $RC(p)$, the diameter of an accelerated counter system is independent of parameters and is less than or equal to $|E| \cdot (|C| + 1) + |C|:

- $|E|$ is the number of edges in TA (self-loops excluded).
- $|C|$ is the number of edge conditions in TA that can be unlocked (locked) by an edge appearing later (resp. earlier) in the control flow, or by a parallel edge.

In our example:

$|E| = 4, |C| = 1.$

Thus,

$d \leq 9.$
Completeness threshold for bounded model checking

Fix a threshold automaton TA and a size function \( N \).

Theorem

For each \( p \) with \( RC(p) \), the diameter of an accelerated counter system is independent of parameters and is less than or equal to \(|E| \cdot (|C| + 1) + |C|:\)

- \(|E|\) is the number of edges in TA (self-loops excluded).
- \(|C|\) is the number of edge conditions in TA that can be unlocked (locked) by an edge appearing later (resp. earlier) in the control flow, or by a parallel edge.

In our example:

\(|E| = 4, |C| = 1.\)

Thus, \( d \leq 9.\)
Can we reach the bound with NuSMV?

- Toy example reached bound at 27 steps.
- Folklore RB reached completeness bound at 10 steps.
- Consistent RB reached bound at 90 steps.
- ABA case 1 reached completeness bound at 1,758 steps.
- ABA case 2 reached completeness bound at 6,620 steps.
- CBC case 1 reached bound at 612 steps.
- CBC case 2 reached completeness bound at 8,720 steps.

Timeout in abstraction refinement: NBAC (13200) and NBACC (16500).
The tool (source code in OCaml), the code of the distributed algorithms in Parametric Promela, and a virtual machine with full setup are available at: http://forsyte.at/software/bymc
Related work: PV of FTDAs

Regular model checking of fault-tolerant distributed protocols:

[Fisman, Kupferman, Lustig 2008]

- “First-shot” theoretical framework.
- No guards like \( x \geq t + 1 \), only \( x \geq 1 \).
- No implementation.
- Manual analysis applied to folklore broadcast (crash faults).
Related work: PV of FTDAs

Regular model checking of fault-tolerant distributed protocols:

[Fişman, Kupferman, Lustig 2008]

- “First-shot” theoretical framework.
- No guards like $x \geq t + 1$, only $x \geq 1$.
- No implementation.
- Manual analysis applied to folklore broadcast (crash faults).

Backward reachability using SMT with arrays:

[Alberti, Ghilardi, Pagani, Ranise, Rossi 2010-2012]

- Implementation.
- Experiments on Chandra-Toueg 1990.
- No resilience conditions like $n > 3t$.
- Safety only.
## Our current work

<table>
<thead>
<tr>
<th></th>
<th>Discrete synchronous</th>
<th>Discrete partially synchronous</th>
<th>Discrete asynchronous</th>
<th>Continuous synchronous</th>
<th>Continuous partially synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>One instance/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>finite payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Many inst./</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>finite payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Many inst./</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unbounded</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Messages with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**one-shot broadcast, c.b.consensus**

- One-shot broadcast, c.b.consensus
- Core of \{ST87, BT87, CT96\}
- MA06 (common), MR04 (binary)
Future work: threshold guards + orthogonal features

Discrete synchronous

Discrete partially synchronous

Discrete asynchronous

Continuous synchronous

Continuous partially synchronous

One instance/finite payload

Many inst./finite payload

Many inst./unbounded payload

Messages with reals

one-shot broadcast, c.b.consensus

core of \{ST87, BT87, CT96\}, MA06 (common), MR04 (binary)

DHM12

CT96
(failure detector)

Broadcast

ST87, BT87, CT96, with failure-detectors

DLS86, MA06, L98 (Paxos)

approx. agreement

DLPSW86

ST87 (JACM)

clock sync

ELPS13

FSFK06

WS07

WS09

Josef Widder (www.forsyte.at)
Thank you!

[ http://forsyte.at/software/bymc ]
Abstract operations

Concrete: $t + 1 \leq x$

Abstract: $I_0, I_1, I_{t+1}, I_{n-t}$
Concrete $t + 1 \leq x$ is abstracted as $x = I_{t+1} \lor x = I_{n-t}$.
Abstract operations

Concrete: \( t + 1 \), \( n - t \)

Abstract: \( I_0 \), \( I_1 \), \( I_{t+1} \), \( I_{n-t} \)

Concrete \( t + 1 \leq x \) is abstracted as \( x = I_{t+1} \lor x = I_{n-t} \).

Concrete \( x' = x + 1 \),
Abstract operations

Concrete: $t + 1 \leq x$ is abstracted as $x = I_{t+1} \lor x = I_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$x = I_0 \land x' = I_1 \ldots$
Abstract operations

Concrete: $t + 1 \leq x$ is abstracted as $x = I_{t+1} \lor x = I_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$$
\begin{align*}
  x &= I_0 \land x' = I_1 \\
  \lor x &= I_1 \land (x' = I_1 \lor x' = I_{t+1}) \ldots
\end{align*}
$$
Abstract operations

Concrete $t + 1 \leq x$ is abstracted as $x = I_{t+1} \lor x = I_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$x = I_0 \land x' = I_1$

$\lor x = I_1 \land (x' = I_1 \lor x' = I_{t+1})$

$\lor x = I_{t+1} \land (x' = I_{t+1} \lor x' = I_{n-t}) \ldots$
Abstract operations

Concrete $t + 1 \leq x$ is abstracted as $x = I_{t+1} \lor x = I_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$x = I_0 \land x' = I_1$
$\lor x = I_1 \land (x' = I_1 \lor x' = I_{t+1})$
$\lor x = I_{t+1} \land (x' = I_{t+1} \lor x' = I_{n-t})$
$\lor x = I_{n-t} \land x' = I_{n-t}$
Parametric abst. refinement — uniformly spurious paths

Classical CEGAR:

Concrete $n_2, t_2, f_2$

Abstract $\cdots$

Concrete $n_1, t_1, f_1$
Parametric abst. refinement — uniformly spurious paths

Classical CEGAR:

Our case: