SimGrid MC
Verification Support for a Multi-API Simulation Platform

Stephan Merz\textsuperscript{1}  Martin Quinson\textsuperscript{2}  Cristian Rosa\textsuperscript{2}

\textsuperscript{1}INRIA Research Center Nancy, France
\textsuperscript{2}Université Henri Poincaré Nancy 1, Nancy, France

FORTE
08/06/2011
Motivation

- Distributed Systems pose a development challenge:
  - lack of a shared clock
  - lack of a global view of the state
  - non-determinism
  - unadapted programming languages

- Verifiy distributed message-passing systems
- We want to study actual implementations
Comparison of methodologies to study distributed systems:

<table>
<thead>
<tr>
<th></th>
<th>Execution</th>
<th>Simulation</th>
<th>Proofs</th>
<th>Model checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Assessment</td>
<td>😞</td>
<td>😞😊</td>
<td>😞</td>
<td>😞</td>
</tr>
<tr>
<td>Experimental Bias</td>
<td>😞😊</td>
<td>😞</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Experimental Control</td>
<td>😞</td>
<td>😞😊</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Correctess Verification</td>
<td>😞😊</td>
<td>😞</td>
<td>😞😊</td>
<td>😞</td>
</tr>
<tr>
<td>Ease of use</td>
<td>😞</td>
<td>😞😊</td>
<td>😞</td>
<td>😞😊</td>
</tr>
</tbody>
</table>

Simulation and Model Checking complement each other:
- Simulation to assess the performance
- Model Checking to verify correctness
- Both run automatically
- No expert users
- Simulators and model checkers often use different models
Studying Distributed Systems

Comparison of methodologies to study distributed systems:

<table>
<thead>
<tr>
<th></th>
<th>Execution</th>
<th>Simulation</th>
<th>Proofs</th>
<th>Model Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Assessment</td>
<td></td>
<td>☻☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>Experimental Bias</td>
<td>☻☻</td>
<td>☻</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Experimental Control</td>
<td>☻</td>
<td>☻☻</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Correctess Verification</td>
<td>☻☻</td>
<td>☻</td>
<td>☻☻</td>
<td>☻</td>
</tr>
<tr>
<td>Ease of use</td>
<td>☻</td>
<td>☻☻</td>
<td>☻</td>
<td>☻☻</td>
</tr>
</tbody>
</table>

Simulation and Model Checking complement each other:
- Simulation to assess the performance
- Model Checking to verify correctness
- Both run automatically
- No expert users
Comparison of methodologies to study distributed systems:

<table>
<thead>
<tr>
<th></th>
<th>Execution</th>
<th>Simulation</th>
<th>Proofs</th>
<th>Model checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Assessment</td>
<td></td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>Experimental Bias</td>
<td>☻</td>
<td>☻</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Experimental Control</td>
<td>☻</td>
<td>☻</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Correctess Verification</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>Ease of use</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
</tbody>
</table>

Simulation and Model Checking complement each other:

- Simulation to assess the performance
- Model Checking to verify correctness
- Both run automatically
- No expert users

Simulators and model checkers often use different models
Model Checking vs. Simulation

Distributed System State Space

- Model Checking explores all (relevant) behavior of the model
- Simulation explores one trace subject to the platform’s restrictions
Model Checking Vs. Simulation

- Model Checking explores all (relevant) behavior of the model
- Simulation explores one trace subject to the platform’s restrictions
Model Checking Vs. Simulation

Distributed System State Space

- Model Checking explores all (relevant) behavior of the model
- Simulation explores one trace subject to the platform’s restrictions
Contributions

The contributions of this article are:

- SimGrid MC: an extension of the SimGrid simulation framework for model checking distributed applications.
- SimGrid MC verify actual implementations
- We present the integration work:
  - Changes to the simulator main loop
  - Simulation and model checking shares many abstractions
- We explain how we implemented DPOR to support different communication APIs
The SimGrid Framework is a collection of tools for the simulation of distributed computer systems.

It uses a discrete-event simulation model (simulated time advances triggered by actions)

Experimental Work-flow:

- Distributed System Implementation
- Resource Models
- Experimental Setup
- Scientific Results
The SimGrid Framework 2

Main loop of SimGrid

- $a$, $b$, $c$, $d$ are communication actions
- they are intercepted to introduce the delays of the platform
- they are the only way to affect the shared state
Main loop of SimGrid

- $a$, $b$, $c$, $d$ are communication actions
- they are intercepted to introduce the delays of the platform
- they are the only way to affect the shared state

The simulator already provides some needed abstractions:
- Controlled execution environment (processes, scheduling, etc).
- Transition detection (communication interception).
1 Introduction
- Motivation
- Model Checking
- The SimGrid Framework 1
- Shared Abstractions

2 SimGrid MC
- Architecture
- DPOR
- The Formal Model

3 Evaluation
- SMPI
- CHORD

4 Conclusion and Future Work
Main characteristics of SimGrid MC:

- Explicit-state exploration (with a depth bound)
- It actually executes the code
- No visited state storing nor hashing (stateless MC)
- Replay based roll-backs
- Properties expressed as assertions (safety only...for now)
- Dynamic Partial-order Reductions to cope with state space explosion
Dynamic Partial-order Reductions

Depth-first search state exploration algorithm:

A

B

C

It's another serialization of the same partial-order!
Dynamic Partial-order Reductions

- Depth-first search state exploration algorithm:

A

\[ \begin{array}{c}
\text{a}_0 \\
\text{s}_a \\
\text{a}_1 \\
\text{l}_a \\
\text{a}_2 \\
\end{array} \]

B

\[ \begin{array}{c}
\text{b}_0 \\
\text{s}_b \\
\text{b}_1 \\
\text{l}_b \\
\text{b}_2 \\
\end{array} \]

C

\[ \begin{array}{c}
\text{c}_0 \\
\text{c}_1 \\
\text{c}_2 \\
\text{a}_0 \\
\text{a}_1 \\
\text{a}_2 \\
\end{array} \]

It's another serialization of the same partial-order!
Dynamic Partial-order Reductions

A
\[ a_0 \quad s_a \quad a_1 \quad l_a \quad a_2 \]

B
\[ b_0 \quad s_b \quad b_1 \quad l_b \quad b_2 \]

C
\[ c_0 \quad c_1 \quad c_2 \]

Depth-first search state exploration algorithm:

\[ a_0 \quad b_0 \quad c_0 \quad a_0 \quad b_1 \quad c_0 \quad a_0 \quad b_1 \quad c_1 \quad a_1 \quad b_1 \quad c_1 \quad a_1 \quad b_2 \quad c_2 \quad a_1 \quad b_2 \quad c_2 \quad a_2 \quad b_2 \quad c_2 \]

\[ s_b \quad r \quad s_a \quad r \quad l_a \quad l_b \]

It’s another serialization of the same partial-order!
Dynamic Partial-order Reductions 1

Depth-first search state exploration algorithm:

It's another serialization of the same partial-order!
What are the transitions that we should interleave?

or equivalently ...

How do we generate a serialization of a different partial-order?
What are the transitions that we should interleave?

or equivalently ...

How do we generate a serialization of a different partial-order?

Interleaving dependent transitions!

\[ D(t_i, t_j) = \neg I(t_i, t_j) \]
How do we get the predicate $D$?

- Using the semantics of the transitions
- Proof of "independence theorems" for each pair of transitions
- The $I$ predicate is the disjunction of these cases

$$I(t_i, t_j) = (t_i = A \land t_j = B) \lor \ldots$$
Computing D Efficiently

How do we get the predicate $D$?

- Using the semantics of the transitions
- Proof of "independence theorems" for each pair of transitions
- The $I$ predicate is the disjunction of these cases

$$I(t_i, t_j) = (t_i = A \land t_j = B) \lor \ldots$$

- It can be over-approximated by a $D'$ such that

$$D(A, B) \Rightarrow D'(A, B)$$

If we don't know if $I(t_i, t_j)$ we assume $D'(t_i, t_j)$ (for soundness).
No communication API has a formal semantics. Manual specification is required based on:

- informal API references
- experiments
- user experience

It is a tedious and time consuming job (should be done for each API)
No communication API has a formal semantics. Manual specification is required based on:
- informal API references
- experiments
- user experience

It is a tedious and time consuming job (should be done for each API)

Our solution:
- A core set of four basic networking primitives
- User-level APIs on top of this
- Formal specification in of it’s semantics
- Theorems of independence between certain primitives
- State-space exploration at primitives’ level
The communication model is based on mailboxes:

- processes post send/receive request into mailboxes
- requests are queued and matched in FIFO order
The communication model is based on mailboxes:
- processes post send/receive request into mailboxes
- requests are queued and matched in FIFO order

There are four primitives:
- **Send** – asynchronous send
- **Recv** – asynchronous receive
- **WaitAny** – block until completion of a communication
- **TestAny** – test for completion without blocking
Example

```c
if (rank % 3 == 0) {
    MPI_Recv(&val1, MPI_ANY_SOURCE);
    MPI_Recv(&val2, MPI_ANY_SOURCE);
    MC_assert(val1 > rank);
    MC_assert(val2 > rank);
} else {
    MPI_Send(&rank, (rank / 3) * 3);
}
```

<table>
<thead>
<tr>
<th>#P</th>
<th>DFS</th>
<th>DPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Time</td>
</tr>
<tr>
<td>3</td>
<td>520</td>
<td>0.247 s</td>
</tr>
<tr>
<td>6</td>
<td>&gt;10560579</td>
<td>&gt;1 h</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
What is Chord?

- Chord is a P2P lookup protocol
- Designed for scalability
- Nodes might join/leave
What is Chord?
- Chord is a P2P lookup protocol
- Designed for scalability
- Nodes might join/leave

Implementation of Chord in SimGrid
- 563 lines of C (MSG interface)
- 2 millions node in simulation
- spotted a bug in big instances
What is Chord?
- Chord is a P2P lookup protocol
- Designed for scalability
- Nodes might join/leave

Implementation of Chord in SimGrid
- 563 lines of C (MSG interface)
- 2 millions node in simulation
- Spotted a bug in big instances

SimGrid MC with two nodes:
- DFS: 15600 states - 24s
- DPOR: 478 states - 1s
- Simple Counter-example!
Conclusions and Future Work

Conclusions:

- Introduced SimGrid MC a model checker for distributed C programs
- Allows to both simulate and verify the programs without modifications
- The integration has been conceptually simple
- DPOR exploration with support for multiple APIs
- Capable of finding bugs in small MPI examples and in a more complex Chord implementation

Future Work:

- Implement and evaluate a stateful exploration
- Experiment with a hybrid roll-back mechanism (checkpoint + replay)
- Add support for liveness properties verification
- Do simulation and model checking at the same time

Merz, Quinson, Rosa (INRIA, UHP Nancy 1)
SimGrid MC
FORTE 08/06/2011
Conclusions and Future Work

Conclusions:

- Introduced SimGrid MC a model checker for distributed C programs
- Allows to both simulate and verify the programs without modifications
- The integration has been conceptually simple
- DPOR exploration with support for multiple APIs
- Capable of finding bugs in small MPI examples and in a more complex Chord implementation

Future Work:

- Implement and evaluate a statefull exploration
- Experiment with an hybrid roll-back mechanism (checkpoint + replay)
- Add support for liveness properties verification
- Do simulation and model checking at the same time