

Performance & Correctness Assessment of Distributed Systems

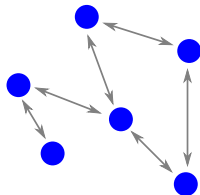
PhD Defense of **Cristian Rosa**

Université Henri Poincaré – Nancy 1, France

24/10/2011

Distributed System

A system that consists of multiple autonomous computing entities that interact towards the solution of a common goal.



Some Examples:

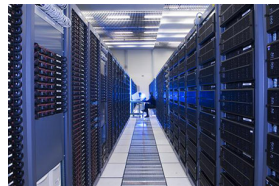
- Facebook: 500 Millions of users
- eBay: idem Facebook + Money
- eMule, BitTorrent, Amazon

Distributed Systems are critical to many applications!

Complexity of Distributed Systems

• Grid Computing

- Infrastructure for computational science
- Heterogeneous computing resources, static network topology
- Main issue: process as much jobs as possible
- Example: LHC Computing Grid – 500K cores, 140 computing centers in 35 countries



LHC Computing Grid

• Peer-to-Peer Systems

- Exploit resources at network edges
- Heterogeneous computing resources, dynamic network topology
- Main issue: deal with intermittent connectivity, anonymity
- Example: BitTorrent, SETI@Home



Peer-to-Peer Network

Challenges of Distributed Systems

- **Lack of knowledge about the global state**
 \leadsto control decisions based only on local knowledge
- **Lack of common time reference**
 \leadsto impossible to order the events of different entities
- **Non-determinism**
 \leadsto evolution of all non-local state impossible to predict

In general distributed systems are badly understood!

Distributed Systems characteristics are hard to assess:

- **Performance**

Must maximize it, but definition differs between systems

- **Correctness**

Hard to find and reproduce bugs, lack of guarantees

- **Theoretical approach**

- 😊 absolute answer
- 😞 often simplistic, time consuming, experienced users

- **Real executions**

- 😊 accurate, real experimentation bias
- 😞 difficult to instrument, limited to a few scenarios

- **Simulations**

- 😊 relative simple to use, many scenarios, fast
- 😞 lack of real experimentation effects, requires validated models

- **Direct Experimentation**

- 😊 no false positives
- 😞 very limited, bugs hard to find and reproduce, difficult to instrument

- **Proofs**

- 😊 complete guarantee of correctness independent of system size
- 😞 non automatic, time consuming, experienced users

- **Model Checking**

- 😊 automatic, relatively simple to use, counter-examples
- 😞 state spaces grows exponentially with system size

Comparison of Methodologies

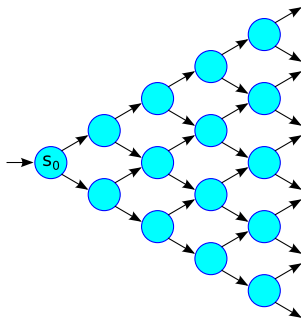
	Execution	Simulation	Proofs	Model checking
Performance Assessment	😊	😊😊	😞	😞
Experimental Bias	😊😊	😊	n/a	n/a
Experimental Control	😞	😊😊	n/a	n/a
Correctness Verification	😞😞	😞	😊😊	😊
Ease of use	😞	😊😊	😞	😊😊

Simulation and Model Checking complement each other:

- Simulation to assess the performance
- Model Checking to verify correctness
- Both run automatically
- Low usability barrier

Often, simulators and model checkers require different system descriptions

Model Checking Versus Simulation



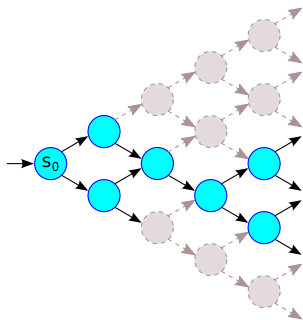
Model Checking idea:

- Exhaustive exploration of state space
- Check validity of every state

In a distributed setting:

- Run all interleavings of communications
- ~ interception of communication events
- ~ control the events' happening ordering

Model Checking Versus Simulation



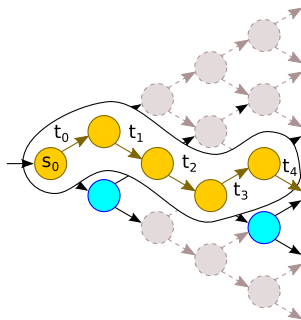
Simulation idea:

- The platform is an additional parameter
- Compute a single run of the system

In a distributed setting:

- Always the same trace with timings
- ~ interception of communications events
- ~ delay the events according to the models

Model Checking Versus Simulation



Simulation idea:

- The platform is an additional parameter
- Compute a single run of the system

In a distributed setting:

- Always the same trace with timings
- ~ interception of communications events
- ~ delay the events according to the models

Objectives and Contributions of the Thesis

Objective

Develop the theory and tools for the efficient performance and correctness assessment of distributed systems.

Approach

Make model checking possible in the SimGrid simulation framework.
Not reinvent the wheel: SimGrid is fast, scalable, and validated.

Contributions

- Correctness assessment:
 - SimGridMC: a dynamic verification tool for distributed systems
 - Custom reduction algorithm to deal with state space explosion
- Performance assessment:
 - Parallelization of the simulation loop for CPU bound simulations
 - Criteria to estimate the potential benefit of parallelism

- 1 Introduction
- 2 Bridging Simulation and Verification
 - The SimGrid Framework
 - SIMIXv2.0
 - Experiments
- 3 SimGrid MC
 - Architecture
 - Coping With The State Explosion
 - Experiments
- 4 Parallel Execution
 - Architecture
 - Cost Analysis
 - Experiments
- 5 Conclusion and Future Work

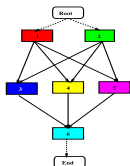
The SimGrid Framework

A collection of tools for the simulation of distributed computer systems

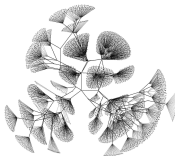
Main characteristics:

- Designed as a scientific measurement tool (validated models)
- It simulates real programs (written in C and Java among others)

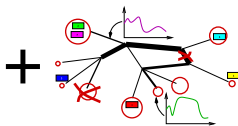
Experimental workflow:



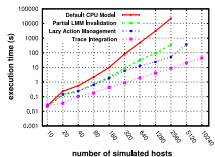
Distributed System
Implementation



Resource Models



Experimental Setup



Scientific Results

A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
 $P_{time} \leftarrow \{P_1, P_2\}$ 
while  $P_{time} \neq \emptyset$  do
  schedule( $P_{time}$ )
  time  $\leftarrow$  solve(&done_actions)
   $P_{time} \leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop

A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
 $P_{time} \leftarrow \{P_1, P_2\}$ 
while  $P_{time} \neq \emptyset$  do
  schedule( $P_{time}$ )
  time  $\leftarrow$  solve(&done_actions)
   $P_{time} \leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



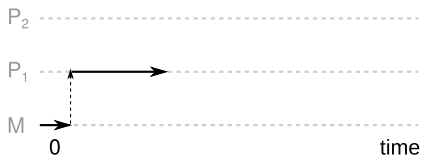
A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
 $P_{time} \leftarrow \{P_1, P_2\}$ 
while  $P_{time} \neq \emptyset$  do
  schedule( $P_{time}$ )
  time  $\leftarrow$  solve(&done_actions)
   $P_{time} \leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



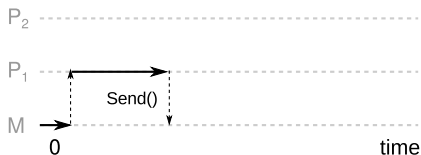
A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
Ptime  $\leftarrow$  {P1, P2}
while Ptime  $\neq$   $\emptyset$  do
  schedule(Ptime)
  time  $\leftarrow$  solve(&done_actions)
  Ptime  $\leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



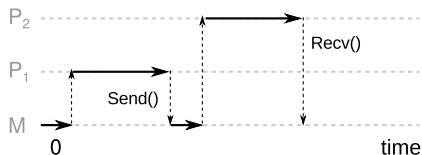
A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
Ptime  $\leftarrow$  {P1, P2}
while Ptime  $\neq$   $\emptyset$  do
  schedule(Ptime)
  time  $\leftarrow$  solve(&done_actions)
  Ptime  $\leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



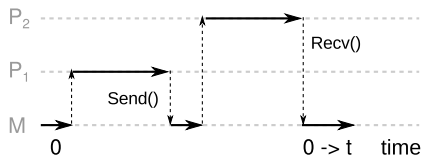
A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
 $P_{time} \leftarrow \{P_1, P_2\}$ 
while  $P_{time} \neq \emptyset$  do
  schedule( $P_{time}$ )
  time  $\leftarrow$  solve(&done_actions)
   $P_{time} \leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



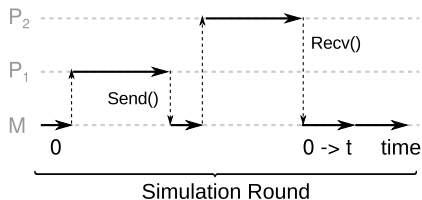
A Simulation Example

```
function P1
  //Compute...
  Send()
  ...
end function
```

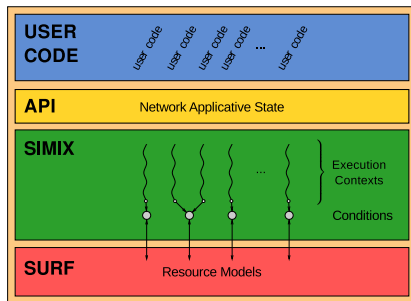
```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
 $P_{time} \leftarrow \{P_1, P_2\}$ 
while  $P_{time} \neq \emptyset$  do
  schedule( $P_{time}$ )
  time  $\leftarrow$  solve(&done_actions)
   $P_{time} \leftarrow$  proc_unblock(done_actions)
end while
```

SimGrid's Main Loop



The Architecture



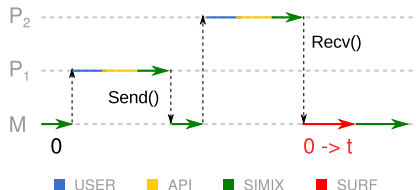
A Simulation Example in More Detail

```
function P1
  //Compute...
  Send()
  ...
end function
```

```
function P2
  //Compute...
  Recv()
  ...
end function
```

```
time  $\leftarrow$  0
P_time  $\leftarrow$  P
while P_time  $\neq$   $\emptyset$  do
  schedule(P_time)
  time  $\leftarrow$  solve(&done_actions)
  P_time  $\leftarrow$  proc_unblock(done_actions)
end while
```

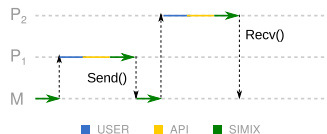
SimGrid's Main Loop



Limitations of the Architecture

This architecture not good enough:

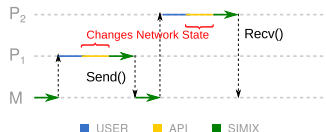
- Late interception of network operations
 \leadsto Lack of control over the network state



Limitations of the Architecture

This architecture not good enough:

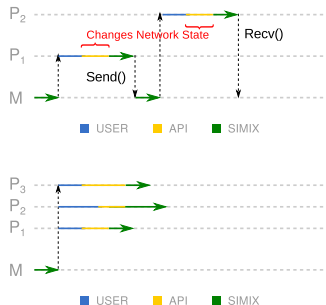
- Late interception of network operations
~> Lack of control over the network state
- User processes modify the shared state
~> Parallel execution hard to achieve
~> Renders reproducibility impossible



Limitations of the Architecture

This architecture not good enough:

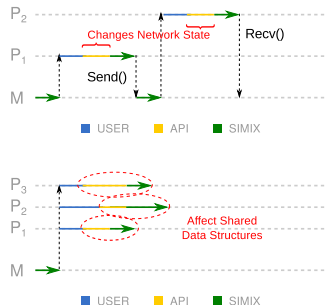
- Late interception of network operations
 \leadsto Lack of control over the network state
- User processes modify the shared state
 \leadsto Parallel execution hard to achieve
 \leadsto Renders reproducibility impossible



Limitations of the Architecture

This architecture not good enough:

- Late interception of network operations
 \leadsto Lack of control over the network state
- User processes modify the shared state
 \leadsto Parallel execution hard to achieve
 \leadsto Renders reproducibility impossible



SIMIXv2.0

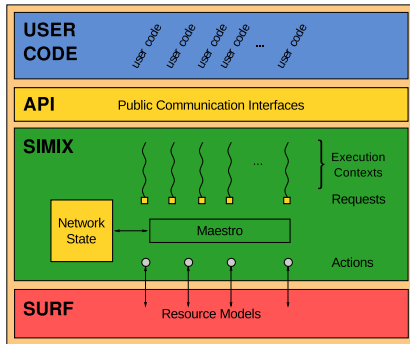
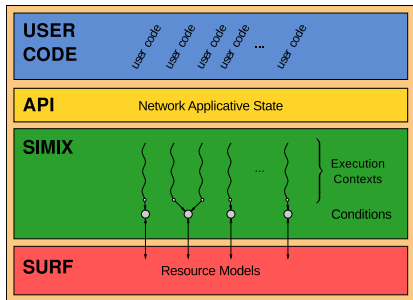
A new virtualization module designed to overcome previous limitations.

(Joint work with Christophe Thiéry).

Inspired by operating system design concepts:

- Strict layered design:
 - Processes, IPC, and synchronization primitives
 - Encapsulated shared state
- System call like interface:
 - Interaction with platform mediated through “requests”
 - The simulator answers the requests

The New Architecture

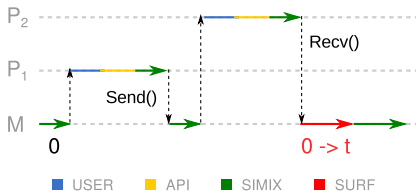


A Simulation Example with SIMIXv2.0

SIMIX Main Loop

```

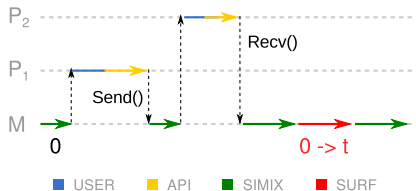
time ← 0
P_time ← P
while P_time ≠ ∅ do
  schedule(P_time)
  time ← solve(&done_actions)
  P_time ← proc_unblock(done_actions)
end while
    
```



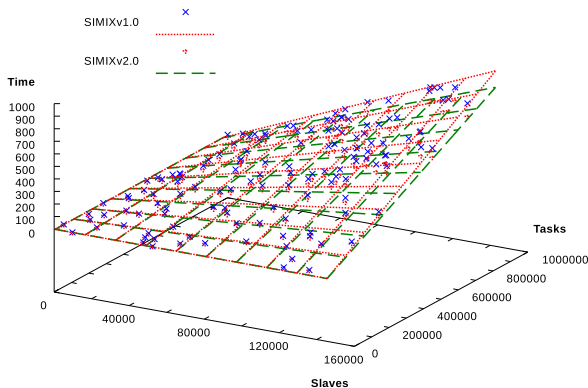
SIMIXv2.0 Main Loop

```

time ← 0
P_time ← {P1, P2}
while P_time ≠ ∅ do
  schedule(P_time)
  handle_requests()
  time ← solve(&done_actions)
  P_time ← proc_unblock(done_actions)
end while
    
```



SIMIXv1.0 versus SIMIXv2.0



Master-Slaves experiment:

- SIMIXv2.0 14% faster on average (with no loses)
- Gains due to: simplification of code, less dynamic data

- 1 Introduction
- 2 Bridging Simulation and Verification
 - The SimGrid Framework
 - SIMIXv2.0
 - Experiments
- 3 SimGrid MC
 - Architecture
 - Coping With The State Explosion
 - Experiments
- 4 Parallel Execution
 - Architecture
 - Cost Analysis
 - Experiments
- 5 Conclusion and Future Work

SimGridMC

A dynamic verification tool for SimGrid programs.

Design Goals:

- Verification of *unmodified* SimGrid programs
- Find bugs triggered by program's nondeterministic behavior
- Designed as a debugging tool
- Capable of handling nontrivial programs
- Simple to use by SimGrid users

An Example of Bug

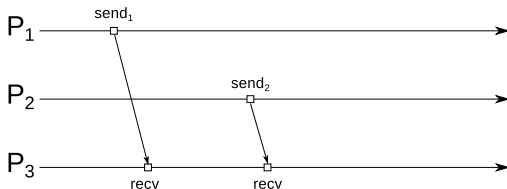
Message Delivery Order Bug

```
P1(){  
    Send(1,P3);  
}  
  
P2(){  
    Send(2,P3);  
}  
  
P3(){  
    Recv(&x,*);  
    Recv(&y,*);  
    ASSERT(x<y);  
}
```

An Example of Bug

Message Delivery Order Bug

```
P1(){  
    Send(1, P3);  
}  
  
P2(){  
    Send(2, P3);  
}  
  
P3(){  
    Recv(&x, *);  
    Recv(&y, *);  
    ASSERT(x < y);  
}
```



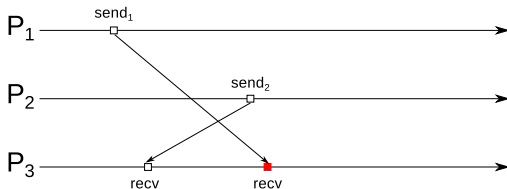
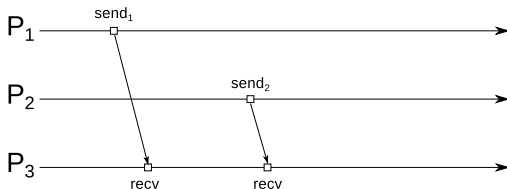
An Example of Bug

Message Delivery Order Bug

```
P1(){  
    Send(1, P3);  
}
```

```
P2(){  
    Send(2, P3);  
}
```

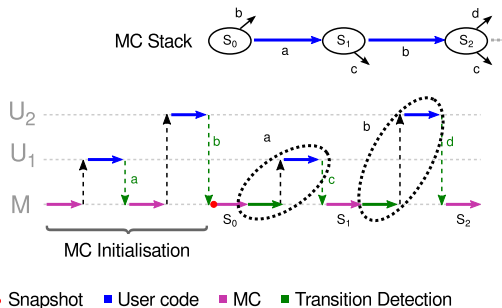
```
P3(){  
    Recv(&x, *);  
    Recv(&y, *);  
    ASSERT(x < y);  
}
```



Main characteristics of SimGridMC:

- Exploration:
 - Explicit-state
 - Verification of local assertions
 - It actually executes the code
- Roll-backs:
 - Stateless approach (replay)
 - No visited state storing nor hashing
- Reduction techniques to cope with state space explosion

The Exploration Loop



Explored Interleavings: $\langle a, b, c, d \rangle, \langle a, b, d, c \rangle, \dots$

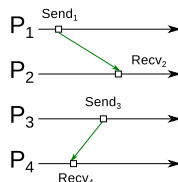
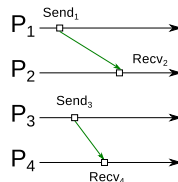
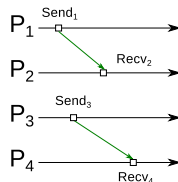
The State Explosion Problem

```
P1(){  
  Send(&x, P2);  
}
```

```
P2(){  
  Recv(&y, P1);  
}
```

```
P3(){  
  Send(&r, P4);  
}
```

```
P4(){  
  Recv(&q, P3);  
}
```

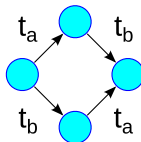


...

They are all the same happened-before relation!

To explore different partial orders we must interleave dependent transitions

$$D(t_a, t_b) = \neg I(t_a, t_b)$$



How do we get the predicate D ?

- Using the semantics of the transitions
- "*Independence theorems*" for each pair of transitions

- No communication API in SimGrid had a formal specification.
 \leadsto Manual specification required (tedious, time consuming)

- No communication API in SimGrid had a formal specification.
 \leadsto Manual specification required (tedious, time consuming)
- The solution explored in this thesis:
 - A core set of four basic networking primitives (SIMIXv2.0 IPC)
 - User-level APIs written on top of these
 - Full formal specification of their semantics (in TLA⁺)
 - Theorems of independence between certain primitives
 - State-space exploration at primitives' level

The Communication Model of the IPC

Communication model based on mailboxes:

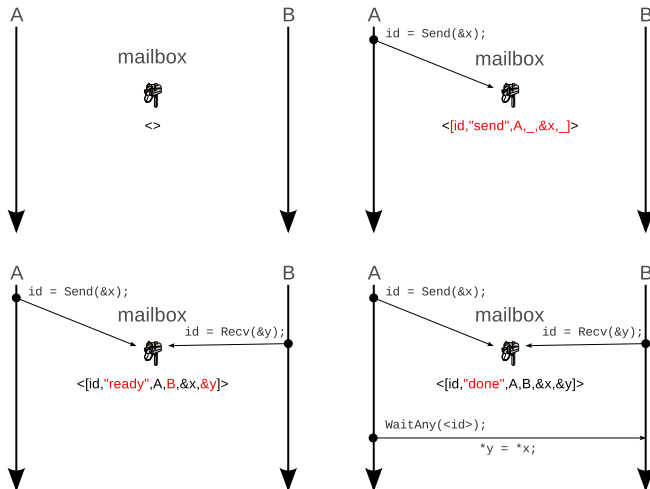
- processes post send/receive request into mailboxes
- requests queued/matched in FIFO order

Four primitives:

- Send – asynchronous send request
- Recv – asynchronous receive request
- WaitAny – block until completion of a communication
- TestAny – test for completion without blocking

Can express large parts of MPI, GRAS (socket), and MSG (CSP)

Semantics of Communication Primitives



Independence Theorems

- 6 theorems of the form:

$$\begin{aligned} I(A, B) &\triangleq \text{ENABLED } A \wedge \text{ENABLED } B \Rightarrow \wedge A \Rightarrow (\text{ENABLED } B)' \\ &\quad \wedge B \Rightarrow (\text{ENABLED } A)' \\ &\quad \wedge A \cdot B \equiv B \cdot A \end{aligned}$$

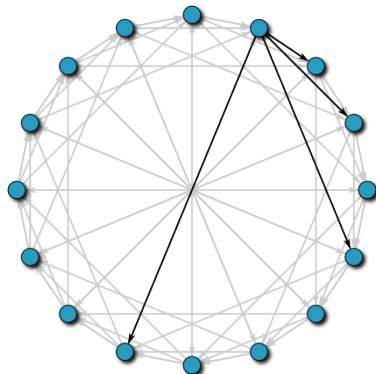
- Proofs expand definitions and use commutativity
- The following actions are independent:
 - Local actions with any other action
 - Send and Recv
 - Two Send or two Recv in different mailboxes
 - Wait or Test for the same communication

Processes multiple of 3 receive a message from their next two successors

```
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);  
}
```

#P	Without reductions			With reductions		
	States	Time	Peak Mem	States	Time	Peak Mem
3	520	0.247 s	23472 kB	72	0.074 s	23472 kB
6	>10560579	>1 h	-	1563	0.595 s	26128 kB
9	-	-	-	32874	14.118 s	29824 kB

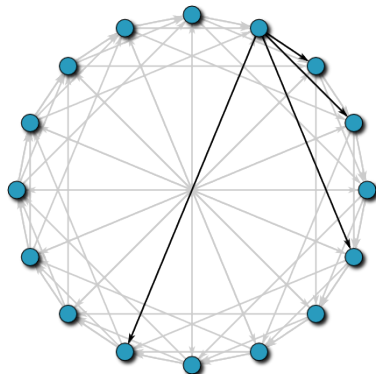
Chord P2P DHT protocol



SimGrid implementation:

- 500 lines of C (MSG interface)
- Spotted a bug in big instances

Chord P2P DHT protocol



SimGrid implementation:

- 500 lines of C (MSG interface)
- Spotted a bug in big instances

SimGrid MC with two nodes:

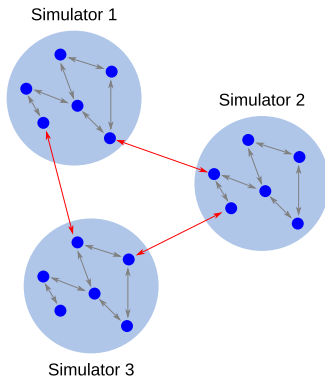
- DFS: 15600 states - 24s
- DPOR: 478 states - 1s
- Simple Counter-example!
- One line fix

- 1 Introduction
- 2 Bridging Simulation and Verification
 - The SimGrid Framework
 - SIMIXv2.0
 - Experiments
- 3 SimGrid MC
 - Architecture
 - Coping With The State Explosion
 - Experiments
- 4 Parallel Execution
 - Architecture
 - Cost Analysis
 - Experiments
- 5 Conclusion and Future Work

Motivation of Parallelization Work

- Scaling-up memory bound simulations; easy \leadsto more RAM
- Speedup CPU bound simulations; difficult
 - Processors increase almost only in parallel power (cores)
 - The simulation problem is really hard to parallelize
- We envision two scenarios:
 - Applications with processes that perform big computations
 - Applications with a large amount of processes

Classical Parallelization Approach



Avoiding out of order events:

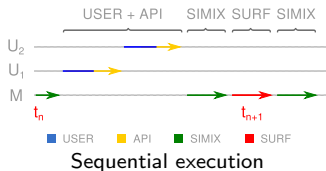
- Conservative: no out of order event can happen
☹ very platform dependent (latency)
- Optimistic: rewind to a consistent state on out of order events
☹ expensive checkpoints

Parallelization of the Simulation Loop

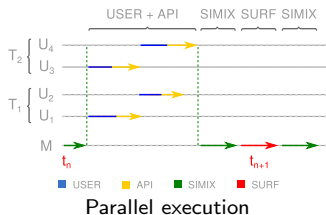
Our Approach

Keep the simulation sequential but parallelize some steps of the main loop

Cost of parallel execution



$$\sum_{t_i} \left(C_{surf}(R, M) + C_{smx}(|P_{t_i}|) + C_{usr}(P_{t_i}) \right)$$



$$\sum_{t_i} \left(C_{surf}(R, M) + C_{smx}(|P_{t_i}|) + C_{thr}(|T|) + \max_{w \in T} (C_{usr}(P_{t_i}^w)) \right)$$

Good Parallelization Scenarios

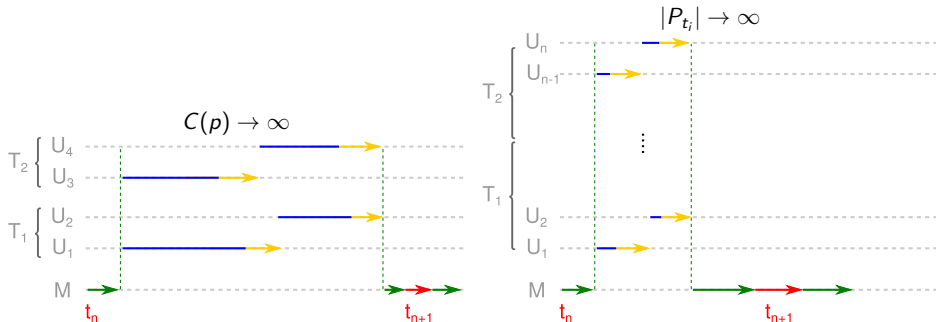
$$K \cdot C_{thr}(|T|) + \max_{w \in T} (C_{usr}(P_{t_i}^w)) < C_{usr}(P_{t_i})$$

Good Parallelization Scenarios

$$K \cdot C_{thr}(|T|) + \max_{w \in T} (C_{usr}(P_{t_i}^w)) < C_{usr}(P_{t_i})$$

This can happen when

$$\sum_{p \in P_{t_i}} C(p) \rightarrow \infty$$



Good scenario: Parallel Matrix Multiplication

- 9 nodes (3x3 grid)
- Matrices of size 1500 (doubles)
- Simulation results (LV08):
 - Sequential execution : 31s
 - Parallel execution (4T): 11s (speedup = x2.8)

Experimental Results II

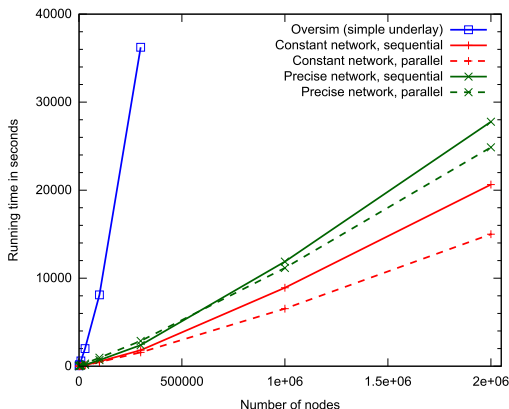
Chord: SimGrid vs. OverSim

300,000 nodes

- OverSim (simple): 10h
- SimGrid (LV08): 38 min

2,000,000 nodes (SG only)

- Seq (LV08): 7h40
- 24T (LV08): 6h55 (x1.30)
- Seq (Const): 5h42
- 24T (Const): 4h (x1.45)



Correctness Assessment:

- Novel approach that integrates a simulator and a model checker
- SimGridMC a model checker for unmodified distributed C programs
- Effective state reduction with support for multiple APIs
- Capable of finding bugs in realistic programs like Chord

Performance Assessment:

- Classical parallelization approaches are not well suited
- Alternative approach: parallelize user processes
- Cost analysis of the approach
- SimGrid is scalable, accurate, and fast

Correctness assessment:

- Implement and evaluate a stateful exploration
- Add support for *liveness* properties verification
- Experiment with a hybrid roll-back mechanism (checkpoint + replay)

Performance assessment:

- Simulation and model checking combined (performance checking)
- Parallelize other steps of the simulation loop
- Refinement of the communication primitives



S. Merz, M. Quinson, and C. Rosa.

Simgrid MC: Verification Support for a Multi-api Simulation Platform.
In 31th Formal Techniques for Networked and Distributed Systems – FORTE 2011, pages 274–288, Reykjavik, Iceland, June 2011.



C. Rosa, S. Merz, and M. Quinson.

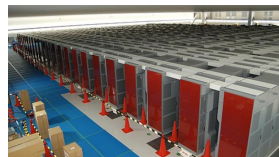
A Simple Model of Communication APIs – Application to Dynamic Partial-order Reduction.

In 10th International Workshop on Automated Verification of Critical Systems – AVOCS 2010, pages 137–151, Dusseldorf, Germany, September 2010.

Taxonomy of Distributed Systems – Part II

- **High Performance Computing**

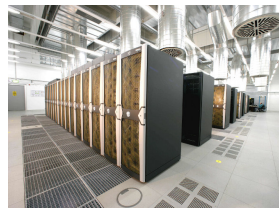
- Lead CS and IT world's research
- Homogeneous nodes with many cores, high-speed local links
- Main issue: do the biggest possible numerical simulations
- Example: K Computer – 548352 Cores, Riken, Japan



K Computer

- **Cloud Computing**

- Large infrastructures underlying commercial Internet
- Heterogeneous computing resources, static network topology
- Main issue: optimize costs, keep up with the load
- Example: Amazon's Cloud



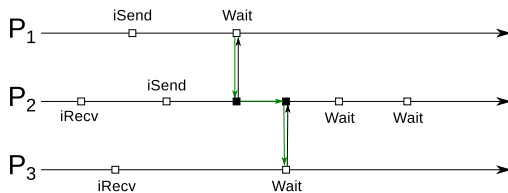
Amazon's Cloud

Asynchronous Communication Bug

```
P1(){  
    c = iSend("ok",P2);  
    Wait(c);  
}  
  
P2(){  
    c1 = iRecv(&buff,P1);  
    c2 = iSend(&buff,P2);  
    Wait(c1);  
    Wait(c2);  
}  
  
P3(){  
    c = iRecv(&buff,P2);  
    Wait(c);  
    ASSERT(buff=="ok");  
}
```

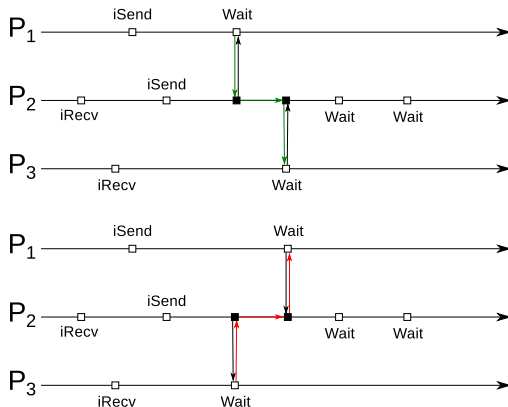
Asynchronous Communication Bug

```
P1(){  
  c = iSend("ok",P2);  
  Wait(c);  
}  
  
P2(){  
  c1 = iRecv(&buff,P1);  
  c2 = iSend(&buff,P3);  
  Wait(c1);  
  Wait(c2);  
}  
  
P3(){  
  c = iRecv(&buff,P2);  
  Wait(c);  
  ASSERT(buff=="ok");  
}
```

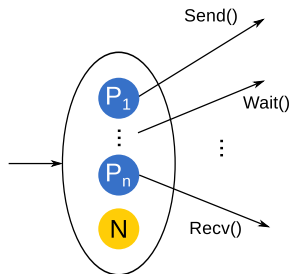


Asynchronous Communication Bug

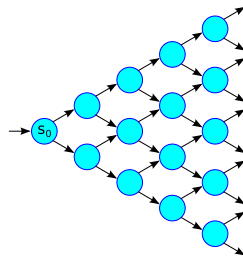
```
P1(){  
    c = iSend("ok",P2);  
    Wait(c);  
}  
  
P2(){  
    c1 = iRecv(&buff,P1);  
    c2 = iSend(&buff,P2);  
    Wait(c1);  
    Wait(c2);  
}  
  
P3(){  
    c = iRecv(&buff,P2);  
    Wait(c);  
    ASSERT(buff=="ok");  
}
```



The Model



State and Transitions

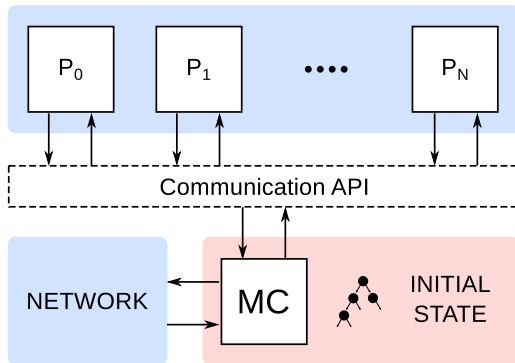


State Space

- The states are the global states of the system
- The transitions are the communication actions
- The exploration consists of interleaving the communication actions

SimGridMC

The Architecture



SimGridMC's Architecture

Approximating Dependency

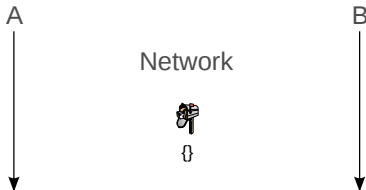
D can be over-approximated by a D' such that

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

If we don't know if $I(t_i, t_j)$ holds we assume $D'(t_i, t_j)$ (for soundness).

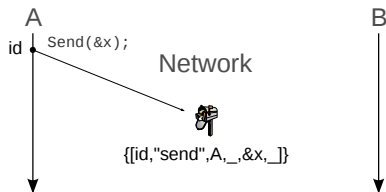
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(\text{Send}(A, rdv_1, \&x, c_1), \text{Recv}(B, rdv_2, \&y, c_2))$$


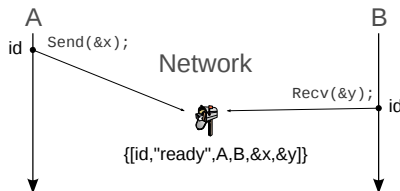
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(\text{Send}(A, rdv_1, \&x, c_1), \text{Recv}(B, rdv_2, \&y, c_2))$$


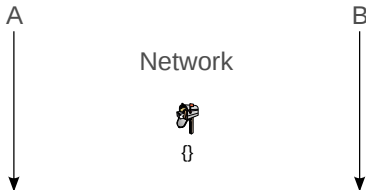
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(Send(A, rdv_1, \&x, c_1), Recv(B, rdv_2, \&y, c_2))$$


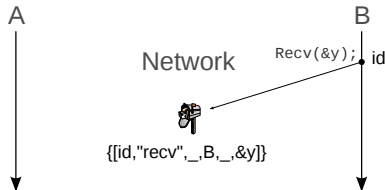
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(Send(A, rdv_1, \&x, c_1), Recv(B, rdv_2, \&y, c_2))$$


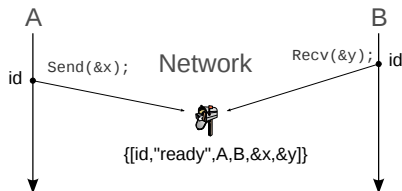
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(\text{Send}(A, rdv_1, \&x, c_1), \text{Recv}(B, rdv_2, \&y, c_2))$$


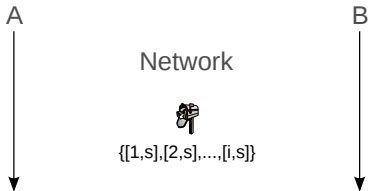
Theorem

Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(\text{Send}(A, rdv_1, \&x, c_1), \text{Recv}(B, rdv_2, \&y, c_2))$$


Theorem

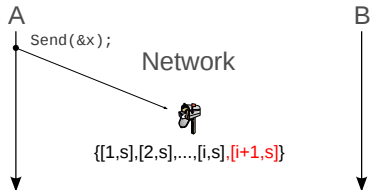
Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(\text{Send}(A, rdv_1, \&x, c_1), \text{Recv}(B, rdv_2, \&y, c_2))$$


Theorem

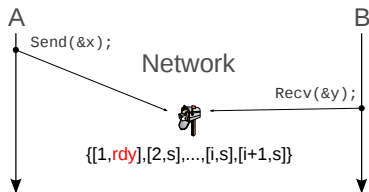
Any two Send and Recv transitions are independent.

$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr :$
 $I(Send(A, rdv_1, \&x, c_1), Recv(B, rdv_2, \&y, c_2))$



Theorem

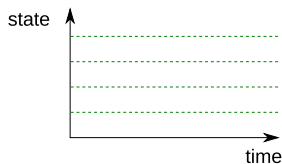
Any two Send and Recv transitions are independent.

$$\forall A, B \in Proc, rdv_1, rdv_2 \in RdV, \&x, \&y \in Addr, c_1, c_2 \in Addr : \\ I(Send(A, rdv_1, \&x, c_1), Recv(B, rdv_2, \&y, c_2))$$


Parallelization of the Simulation Loop

Classical Parallelization Approaches

There are two classical parallelization approaches:



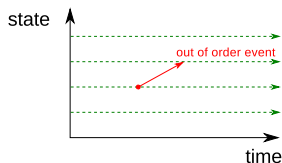
Space Decomposition

- Multiple time lines

Parallelization of the Simulation Loop

Classical Parallelization Approaches

There are two classical parallelization approaches:



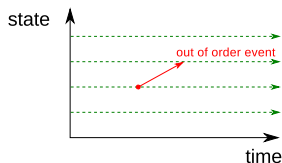
Space Decomposition

- Multiple time lines
- Risk of out of order events
- Conservative: advance when no event out of order can happen
- Optimistic: rewind to a consistent state when out of order events happen

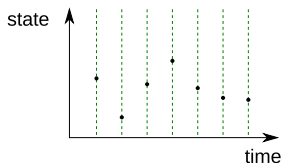
Parallelization of the Simulation Loop

Classical Parallelization Approaches

There are two classical parallelization approaches:



Space Decomposition



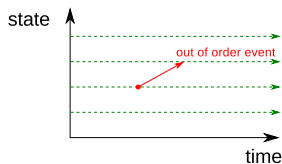
Time Decomposition

- Multiple time lines
 - Risk of out of order events
 - Conservative: advance when no event out of order can happen
 - Optimistic: rewind to a consistent state when out of order events happen
- Time divided in intervals
 - Intervals simulated in parallel

Parallelization of the Simulation Loop

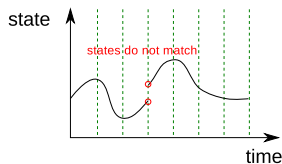
Classical Parallelization Approaches

There are two classical parallelization approaches:



Space Decomposition

- Multiple time lines
- Risk of out of order events
- Conservative: advance when no event out of order can happen
- Optimistic: rewind to a consistent state when out of order events happen

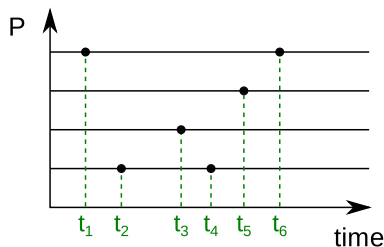


Time Decomposition

- Time divided in intervals
- Intervals simulated in parallel
- Must guess initial states
- Re-computation needed when states do not match

Resolution of the Model

A simulation defines a discretization of the simulated time:



Each event has a timestamp that is computed using the resource models.

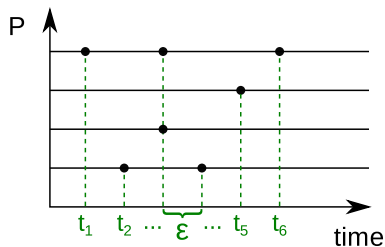
$$T_{M,R} : E \rightarrow [0, t], \text{ with } t \in \mathbb{R}$$

Resolution (ε)

The minimal time increment possible between two timestamps.

Importance of the Model's Resolution

The model resolution ε has an impact on the size of P_{t_i}



Higher resolution

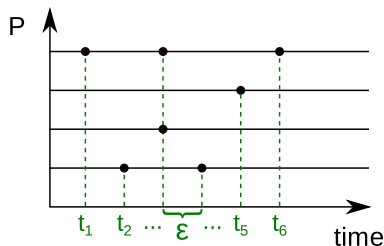
$$|P_{t_1}| = 1, \quad |P_{t_2}| = 1$$

$$|P_{t_3}| = 2, \quad |P_{t_4}| = 1$$

$$|P_{t_5}| = 1, \quad |P_{t_6}| = 1$$

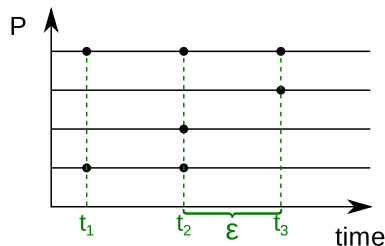
Importance of the Model's Resolution

The model resolution ε has an impact on the size of P_{t_i}



Higher resolution

$$\begin{aligned} |P_{t_1}| &= 1, & |P_{t_2}| &= 1 \\ |P_{t_3}| &= 2, & |P_{t_4}| &= 1 \\ |P_{t_5}| &= 1, & |P_{t_6}| &= 1 \end{aligned}$$



Lower resolution

$$\begin{aligned} |P_{t_1}| &= 2 \\ |P_{t_2}| &= 3 \\ |P_{t_3}| &= 2 \end{aligned}$$

Impact of the resolution ε on $|P_{t_i}|$

- Chord 100,000 nodes
- Simulation of 1000s
- 25,000,000 messages exchanged

ε	10^{-5}	10^{-3}	10^{-1}	Constant network
Average $ P_{t_i} $	10	44	251	7424

Chord: SimGrid vs. OverSim

Chord 2,000,000 nodes

- $\varepsilon = 10^{-1}$
- Sequential execution: 8h15
- Parallel execution: 7h15
- speedup = 1.13 (24 threads)

