Computational Science *of* Computer Systems

Méthodologies d'expérimentation pour l'informatique distribuée à large échelle

Martin Quinson

March 8th, 2013

UNIVERSITÉ DE LORRAINE

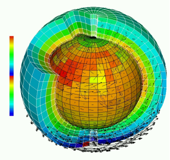
Loria
Laboratoire lorrain de recherche en informatique et ses applications

SIMGRID

What is Science anyway?

Doing Science = Acquiring Knowledge

$$\frac{\partial}{\partial x_j}\left(\frac{\partial \Phi}{\partial x_i}\right) = \frac{\partial}{\partial x_i}\left(\frac{\partial \Phi}{\partial x_j}\right)$$



Experimental Science

- Thousand years ago
- Observations-based
- Can describe
- Prediction tedious

Theoretical Science

- Last few centuries
- Equations-based
- Can understand
- Prediction long

Computational Science

- Nowadays
- Compute-intensive
- Can simulate
- Prediction easier

Prediction is very difficult, especially about the future. – Niels Bohr

Observations still base Science

Space telescope

Large Hadron Collider

Mars Explorer

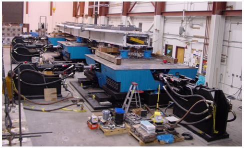
NMR Spectroscope

Synchrotrons

Turntable

Tsunamis

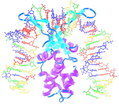
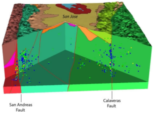
Earthquake vs. Bridge

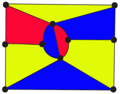
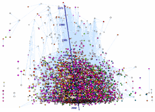
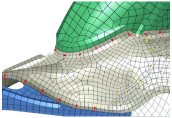
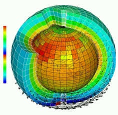


Climate vs. Ecosystems

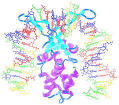
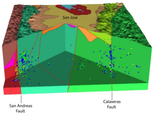
(who said that science is not fun??)

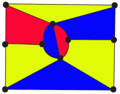
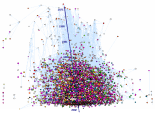
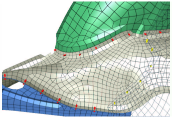
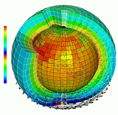
Computational Science

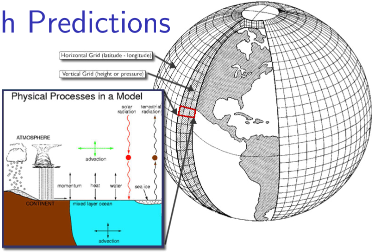
Computational Science

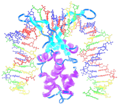





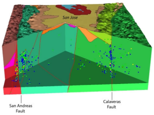
Understanding the Climate Change with Predictions

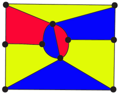


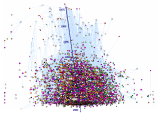


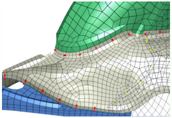
Computational Science

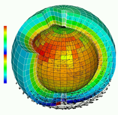




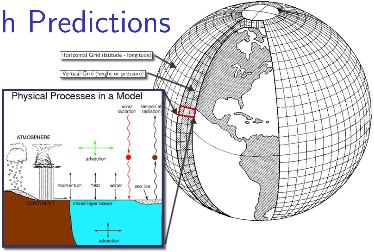




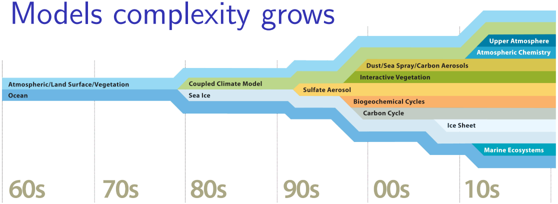




Understanding the Climate Change with Predictions



Models complexity grows



This requires **large** computers

Upscale project:

15,000 computing-years in 2012!

Modern Computers are Large and Complex

Massive Parallelism

- Cannot miniaturize further (atom limit)
- Cannot increase frequency (energy limit)
- Solution: Multiply compute cores!
- Sequoia, second fastest computer: 1,572,864 cores

ExaScale Systems, used in Computational Science

- Systems computing 1 Exaflop per second arrive (with *billions* of cores)
- 1 Exaflop = 10^{18} operations. One million million million operations...
- At humanly doable speed, that requires 10 times the age of the universe
- Each node: 20 millions lines of code ($10\times$ Encyclopedia Britannica)

Other very large computer systems in the wide

- Google computers dissipate 300MW on average (150,000 households, $\frac{1}{3}$ reactor)
- Botnets: BredoLab estimated to control 30 millions of zombie computers
- In addition, these systems are heterogeneous and dynamic

Computational Science **of** Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of large-scale computer systems
- Meta-research on producing scientifically sound results
- Main contribution: SimGrid, a large-scale computer systems simulator

Computational Science of Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of large-scale computer systems
- Meta-research on producing scientifically sound results
- Main contribution: SimGrid, a large-scale computer systems simulator

First title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Computational Science *of* Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of large-scale computer systems
- Meta-research on producing scientifically sound results
- Main contribution: SimGrid, a large-scale computer systems simulator

First title (rejected)

La simulation d'applications pour la recherche
en applications de simulation pour la recherche

Computational Science of Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of **large-scale computer systems**
- Meta-research on **producing scientifically sound results**
- Main contribution: **SimGrid, a large-scale computer systems simulator**

First title (rejected)

Simulating Applications for **Research** in
Simulation Applications for Research

Computational Science *of* Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of **large-scale computer systems**
- Meta-research on **producing scientifically sound results**
- Main contribution: **SimGrid, a large-scale computer systems simulator**

First title (rejected)

Simulating Applications for **Research** in
Simulation Applications for Research

Epistemological Stance

- Empirically consider large-scale computer systems as natural objects
- Eminently artificial artifacts, but complexity reaches “natural” levels
- Other sciences routinely use computers to understand complex systems

Assessing Distributed Applications

Correctness Study $\leadsto$ Formal Methods

- Tests: Unable to provide definitive answers

Performance Study $\leadsto$ Experimentation

- Maths: Often not sufficient to fully understand these systems

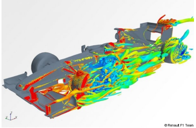
Assessing Distributed Applications

Correctness Study ⇝ Formal Methods

- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study ⇝ Experimentation

- Maths: Often not sufficient to fully understand these systems



Courtesy of Lucas Nussbaum

- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

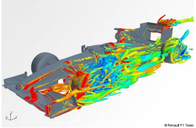
Assessing Distributed Applications

Correctness Study ⇝ Formal Methods

- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study ⇝ Experimentation

- Maths: Often not sufficient to fully understand these systems



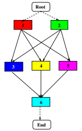
Courtesy of Lucas Nussbaum

- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Emulation: Real applications on Synthetic platforms *(in vitro)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

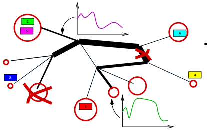
Simulating Distributed Systems

Big Idea: Simulation is the fastest path from idea to scientific results

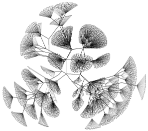
Idea to test — Experimental setup — Simulation Model — Scientific results



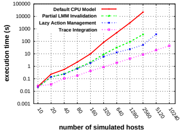
+



+



⇒



Comfort to the user

- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, ignore technical subtleties (for now)

Challenges for the tools

- Validity: Get realistic results (controlled experimental bias)
- Scalability: *Fast enough* and *Big enough*; Tooling: runner, post-processing

Scientific practices sometimes unfortunate in this field

- Experimental settings not detailed enough in literature
- Many short-lived simulators; few sound and established tools

SimGrid: Versatile Simulator of Distributed Apps

Scientific Instrument

- Versatile: Grid, P2P, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Community-driven: 30 contributors (5 not affiliated), 5 contributed tools, GPL

Scientific Object

- Allows comparison of network models on non-trivial applications
- High-Performance Simulation on realistic workload
- Full model checker of distributed applications; Emulator under way

Large Established Project

- Started in 1998; Collab. Loria / Inria Grenoble / CC-IN2P3 / U. Hawaii
- Impact: 120 publications (110 distinct authors, 5 continents), 4 PhD
- Co-leader with A. Legrand (CNRS Grenoble) and F. Suter (CNRS IN2P3)

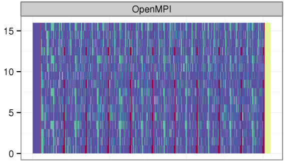
Simulation Validity

SotA: Models in most simulators are either simplistic, wrong or not assessed

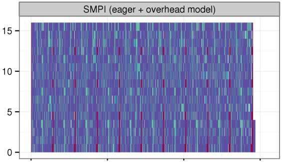
- PeerSim: discrete time, application as automaton;
- GridSim/CloudSim: naive packet level or buggy flow sharing
- OptorSim, GroudSim: documented as wrong on heterogeneous platforms

SimGrid provides several Network Models

- Flow-based: Contention, Slow-start, TCP congestion, Cross-traffic effects
- Constant time: A bit faster, but no hope of realism
- Coordinate-based: Easier to instantiate in P2P scenarios
- Packet-level: NS3 bindings



Real Sweep3D



Simulated Sweep3D

Major Contributions (with many contributors)

1/ **Proto-Emulation:** Assessing Real Applications

- GRAS: Middleware to run simulation prototypes on real platforms
- SMPI: Study real MPI applications within SimGrid

2/ **HPS:** High Performance and Scalable Simulation

- Fast Enough: Innovative PDES; Efficient algorithms and implementations
- Big Enough: Scalable and versatile platform representation

3/ **Formal:** Correctness Studies in SimGrid

- Seamless integration of a complete Model Checker (enforces code invariants)
- Exhaustive reachability analysis, with innovative versatile DPOR technique

Scientific Community Management

- Project Coordinator: 2 ANR projects, 1 regional CPER project (total: 4M€)
- Methodological convergence: Board member of Grid'5000 experimental grid
- Scientific Animation (SimGrid, Grid'5000): 4 summer schools, 3 R&D engineers

\+ leading role in teaching, pedagogical tools, popularization and didactic projects

Major Contributions (with many contributors)

1/ **Proto-Emulation:** Assessing Real Applications

- GRAS: Middleware to run simulation prototypes on real platforms
- SMPI: Study real MPI applications within SimGrid

2/ **HPS:** High Performance and Scalable Simulation

- Fast Enough: Innovative PDES; Efficient algorithms and implementations
- Big Enough: Scalable and versatile platform representation

3/ **Formal:** Correctness Studies in SimGrid

- Seamless integration of a complete Model Checker (enforces code invariants)
- Exhaustive reachability analysis, with innovative versatile DPOR technique

Scientific Community Management

- Project Coordinator: 2 ANR projects, 1 regional CPER project (total: 4M€)
- Methodological convergence: Board member of Grid'5000 experimental grid
- Scientific Animation (SimGrid, Grid'5000): 4 summer schools, 3 R&D engineers

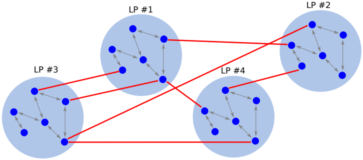
+ leading role in teaching, pedagogical tools, popularization and didactic projects

Parallel Simulation of Discrete Event Systems

- 30 years of literature on efficient Simulation Engines, FES and distribution
- Yet, all DES simulator for P2P were sequential (but dPeerSim)

The dPeerSim attempt

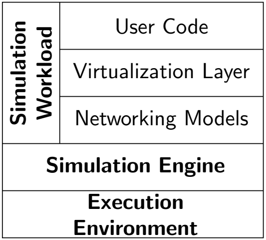
- Distributed implementation of PeerSim
- Classical parallelization: spreads the load over several Logical Processes (LP)



Evaluation

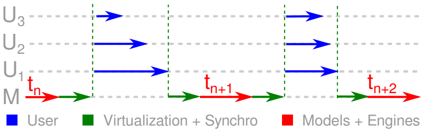
- Uses Chord as a standard workload: e.g. 320,000 nodes $\rightsquigarrow$ 320,000 requests
- Very good speedup results: 4h on 2 LPs $\rightsquigarrow$ 1h on 16 LPs
- But 47s in the original sequential PeerSim (and 5s in precise SimGrid)
- Yet, best known parallelization of DES simulator of P2P systems

New Parallelization Schema for DES



Split at Virtualization, not Simulation Engine

- Virtualization contains threads (user's stack)
- Engine & Models remains sequential



Understanding the trade-off

- Sequential time: $\sum_{SR}(engine + model + virtu + user)$
- Classical schema: $\sum_{SR}\left(\max_{i \in LP}(engine_i + model_i + virtu_i + user_i) + proto\right)$
- Proposed schema: $\sum_{SR}\left(engine + model + \max_{i \in WT}(virtu_i + user_i) + sync\right)$
- Synchronization protocol expensive wrt the engine's load to be distributed

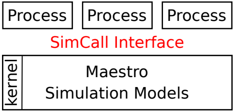
Toward Parallel P2P Simulation in SimGrid

Keep models sequential, execute processes in parallel

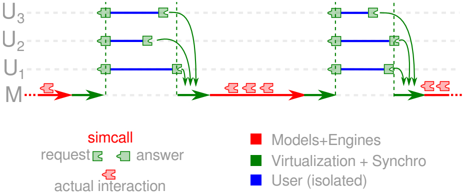
OS-inspired Approach toward Process Separation

- Fine-locking would be difficult, inefficient and would hinder reproducibility
- Mediate any process interactions through simcalls
  (conceptually identical to *syscalls* of real OSes)

Functional View

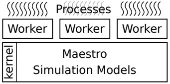


Temporal View

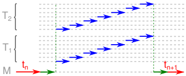


Leveraging Multicores

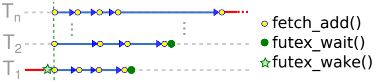
⇒ More processes than cores ⇝ Worker Threads (execute co-routines ;)



Functional View



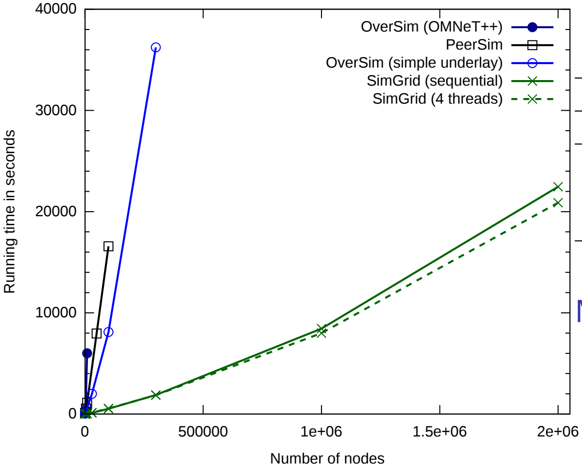
Temporal View



Ideal Algorithm

Sequential Performance in State of the Art

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



Largest simulated scenario

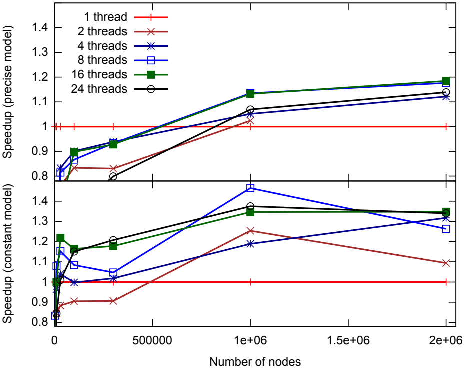
	Size	Time
Omnet++	10k	1h40
PeerSim	100k	4h36
OverSim	300k	10h
SG, precise	10k	130s
	300k	32mn
	2M	6h23
SG, simple	2M	5h30

Memory Usage

- 2M precise nodes: 32 GiB
- That is 18kiB per process (User stack: 12kiB)

Extra complexity of parallel execution don't impact sequential performance

Benefits of the Parallel Execution



- Speedup ($\frac{t_{seq}}{t_{par}}$): up to 45%
- More efficient with simple model:
  - Less work in engine + Amhdal law
- Speedup depends on thread amount
  - 8 threads (of 24 cores) often better
  - Synch costs remain hard to amortize
  - They depend on thread amount

Parallel Efficiency ($\frac{speedup}{\#cores}$) for 2M nodes

Model	4 threads	8 th.	16 th.	24 th.
Precise	0.28	0.15	0.07	0.05
Constant	0.33	0.16	0.08	0.06

- Baaaaad efficiency results
- Remember, P2P and Chord: Worst case scenarios

Yet, first time that Chord's parallel simulation is faster than best known sequential

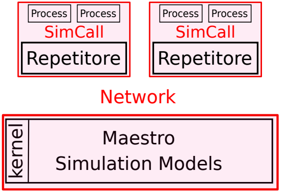
Future Work on HPS

Distributed Simulation toward size

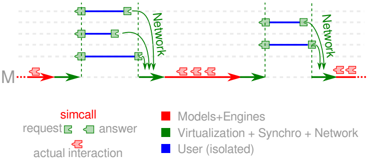
- Leverage the memory of more nodes; Useless in P2P, more adapted to SMPI

Design: split our design under the simcall layer

Functional View



Temporal View



Increase level of parallelism

- Pessimistic execution (as now): efficient for 500,000 processes and more. . .
- Optimistic execution unfeasible because of our complex state
- Vision: realistic execution run optimistically only if it is safe to do so
  Determining independent actions is easy using formal methods

Major Contributions (with many contributors)

1/ **Proto-Emulation:** Assessing Real Applications

- GRAS: Middleware to run simulation prototypes on real platforms
- SMPI: Study real MPI applications within SimGrid

2/ **HPS:** High Performance and Scalable Simulation

- Fast Enough: Innovative PDES; Efficient algorithms and implementations
- Big Enough: Scalable and versatile platform representation

3/ **Formal:** Correctness Studies in SimGrid

- Seamless integration of a complete Model Checker (enforces code invariants)
- Exhaustive reachability analysis, with innovative versatile DPOR technique

Scientific Community Management

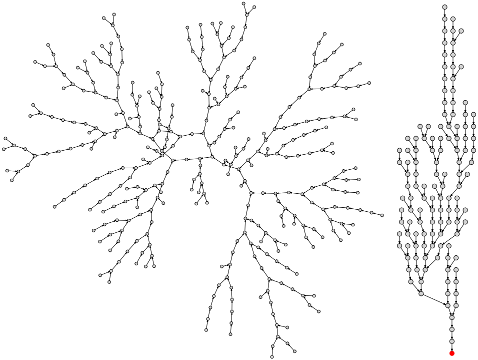
- Project Coordinator: 2 ANR projects, 1 regional CPER project (total: 4M€)
- Methodological convergence: Board member of Grid'5000 experimental grid
- Scientific Animation (SimGrid, Grid'5000): 4 summer schools, 3 R&D engineers

+ leading role in teaching, pedagogical tools, popularization and didactic projects

Exhaustive Testing for Correctness Formal Assesment

Model Checking's Big Idea

- Explore all possible executions of the system
- Actively searching for property violations

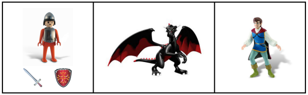


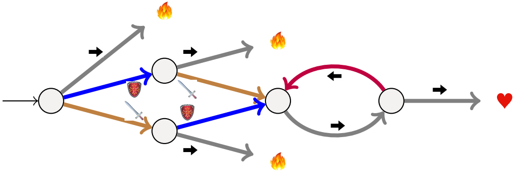
Testing can only prove the presence of bugs. — Dijsktra

well, unless it's exhaustive :)

Model Checking in Wonderland

A warrior seeks her prince.
She can grab 🗡, grab 🛡, move ➡, move ⬅.





Model checking: Actively search for a counter example

- If not found, then the property was true after all
- If found, we got a counter-example (very precious during bug squashing)

Safety Property: $\Box(\neg$ 🔥$)$

- Search an invalidating state
- Exhaustive traversal: property true

Liveness Property: $\Box(($🗡 $\wedge$ 🛡$) \Rightarrow \Diamond$ ♥$)$

- Search a cycle w/ property is false
- Counter-example is infinite

The Problem with Model Checking

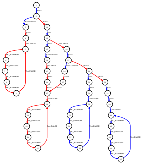
I use programs, not models

- Model-checking usually done on logical models, e.g. expressed with TLA$^+$
- Some technics require the full graph, that I never have

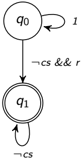
$\Rightarrow$ Explicit exploration of Implicit graph is called Dynamic Verification

Liveness Properties

- Nice properties are liveness ones, not safeties, but that's much harder
- Counter example must be of infinite length, so encoded as Buchi automaton



$\times$



Any process that asks the critical section will get it

- r: request
- cs: critical section
- LTL property: $\Box(r \Rightarrow \Diamond cs)$

State-space Explosion

- Nice problems require $2^{2^{100}}$ years in practice (or more)
- Several reduction technics exists, but preserving cycles is harder

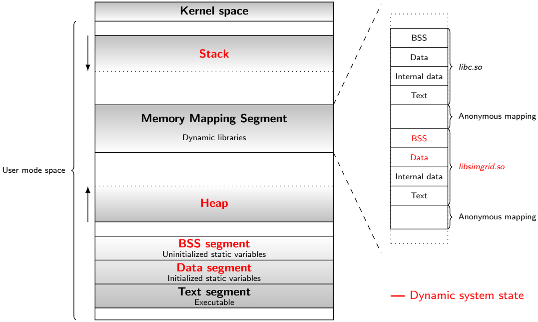
Dynamic Verification in SimGrid

Verifying safety properties

- It works (MSG & SMPI); Reduction with DPOR-based reduction techniques
- Found *wild* bugs in medium-sized programs (Chord protocol)

Verifying liveness properties (ongoing)

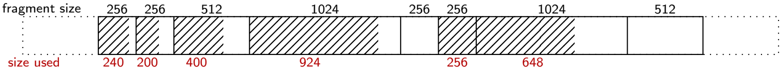
- Problem: detect when the system reenters an (accepting) state
- We need system-level state equality



- Byte-per-byte comparison ineffective
- Lots of false negatives (aka undetected violations)

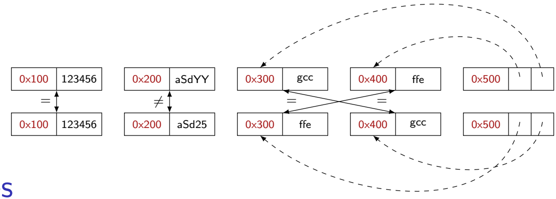
Challenges of System-level State Equality

Over provisioning



Syntactic differences

- In malloc, blocs order can vary without impacting applicative semantic



Padding Bytes

- Data is aligned in memory for efficiency, leaving holes

Irrelevant differences

- Host-related data (pid, files), simulation-related data (time)

Toward Liveness Properties in SimGrid

System Solutions to this Formal Problem

Problem	Heap solution	Stack solution
Over provisioning	Memset 0 + requested size	Stack pointer
Padding bytes	Memset 0	DWARF + libunwind
Irrelevant differences	MC_ignore	DWARF + libunwind
Syntactic differences	Canonicalization	N/A

Current state

- Toy artificial bugs found; Toy property on non-tivial code (NeverJoin in Chord)
- State equality gives a new reduction that works on liveness, too

Future

- MPI3 asynchrone collective operations are a call for semantic bugs
- Assessing properties on communication schema toward easier checkpointing
- Assessing linearizability (service is robust to concurrent usages)
- Explore specific reduction techniques for distributed apps

Take Away Messages

SimGrid will prove helpful to your research

- Versatile: Used in several communities (scheduling, GridRPC, HPC, P2P, Clouds)
- Accurate: Model limits known thanks to validation studies
- Sound: Easy to use, extensible, fast to execute, scalable to death, well tested
- Open: User-community much larger than contributors group; LGPL
- Around since over 10 years, and ready for at least 10 more years

Welcome to the Age of (Sound) Computational Science

- Discover: `http://simgrid.gforge.inria.fr/`
- Learn: 101 tutorials, user manuals and examples
- Join: user mailing list, #simgrid on irc.debian.org
  We even have some open positions ;)

The Computational Science Nightmare

Computational Science is rarely Reproducible!

- Scientific publications must include all information needed for reproduction
- Knowledge is not the finding, but the method. – Boyle

Issue shared with other scientific disciplines

- *Why Most Published Research Findings are False.* Ioannidis, PloS Med, 2005.
- *Reproducibility in Computational and Experimental Maths* workshop, 12/2012

JASA June	Computational Articles	Code Available
1996	9 of 20	0%
2006	33 of 35	9%
2009	32 of 32	16%
2011	29 of 29	21%

V. Stodden

Non-CS major will teach us about Computational Science!

(inspired from Victoria Stodden, Department of Statistics, Columbia University)

Open Science, and CS^2

Required Tools

- Standard tools: Matlab, R in statistics, ...
- Dissemination Platforms: RunMyCode.org
- Workflow Tracking and Research Environments: VisTrails, MyExperiment.org
- Embedded Publishing: Sweeve
- Journal Policy: Things evolve veeeery slowly

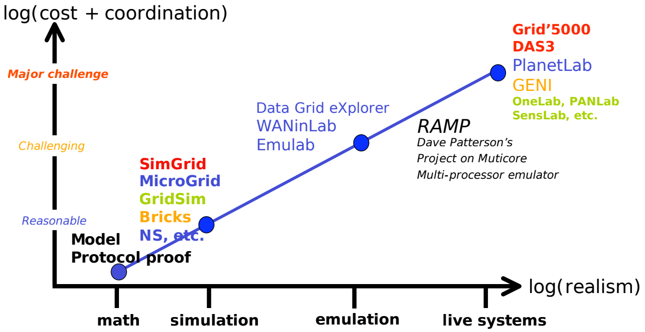
My Research Plan

- SimGrid is a standard tool; use it as troyan to pass best practices along
- Ease experiment packaging and sharing
- Increase associated tools (adaptative runners) to increase the incentive
- Improve our own best practices within the team
- Learn from other disciplines, and build upon this

Conclusion

Scientific Instruments for Distributed Systems

- Common Belief in 2008: Simulation as a toy methodology

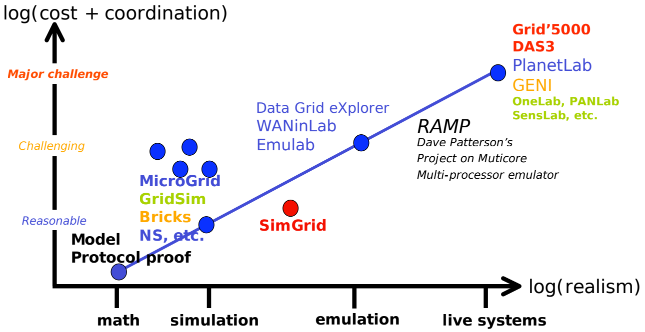


Courtesy of Franck Cappello (Gri5000 keynote @ EGEE, Feb 2008 :)

Conclusion

Scientific Instruments for Distributed Systems

- Common Belief in 2008: Simulation as a toy methodology
- Consensus in 2013: SimGrid as a scientific instrument (w/ Grid'5000)

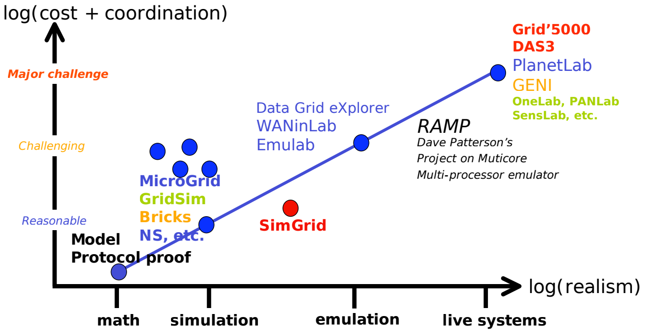


Simulation turned into a reliable scientific instrument!

Conclusion

Scientific Instruments for Distributed Systems

- Common Belief in 2008: Simulation as a toy methodology
- Consensus in 2013: SimGrid as a scientific instrument (w/ Grid'5000)
- Consensus in 2020? We were naïve in 2010, but it works better now



Simulation turned into a reliable scientific instrument!

But there is still a long way to go!

Research Program

Computational Science *of* Computer Systems

pursued convergence of Simulation, Dynamic Verification and Emulation

1/ Modeling of Large-Scale Systems

- Scalability and Accuracy still not enough for Exascale studies
- Semantic modeling of MPI 3.0 collectives (implementation-depend)

2/ Formal Methods for Large-Scale and HPC Systems

- Liveness properties on legacy code (OS-level introspection tooling)
- Domain-specific properties and reduction techniques

3/ Simulation of Real Applications

- OS Virtualization layer for the simulation of legacy code
- Distributed simulation, and increase parallelism in our simulation

4/ Scientific Instrument and Open Science

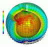
- Produce a *de facto* standard tool, with associated tools
- Foster the emergence of a vivid research community, with best practices

Question slides

What is Science anyway?

Doing Science = Acquiring Knowledge

$$\frac{\partial}{\partial x_j}\left(\frac{\partial \Phi}{\partial x_i}\right) = \frac{\partial}{\partial x_i}\left(\frac{\partial \Phi}{\partial x_j}\right)$$



Experimental Science
- Thousand years ago
- Observations-based
- Can describe
- Prediction tedious

Theoretical Science
- Last few centuries
- Equations-based
- Can understand
- Prediction long

Computational Science
- Nowadays
- Compute-intensive
- Can simulate
- Prediction easier

Prediction is very difficult, especially about the future. – Niels Bohr

Observations still base Science

Space telescope

Large Hadron Collider

Mars Explorer

NMR Spectroscope

Synchrotrons

Turntable

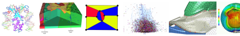
Tsunamis

Earthquake vs. Bridge

Climate vs. Ecosystems

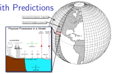

(who said that science is not fun??)

Computational Science

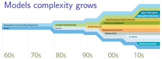


Understanding the Climate Change with Predictions





Models complexity grows



This requires **large** computers

Upscale project:
15,000 computing-years in 2012!

Modern Computers are Large and Complex

Massive Parallelism
- Cannot miniaturize further (atom limit)
- Cannot increase frequency (energy limit)
- Solution: Multiply compute cores!
- Sequoia, second fastest computer: 1,572,864 cores

ExaScale Systems, used in Computational Science
- Systems computing 1 Exaflop per second arrive (with *billions* of cores)
- 1 Exaflop = 10^{18} operations. One million million million operations...
- At humanly doable speed, that requires 10 times the age of the universe
- Each node: 20 millions lines of code (10× Encyclopedia Britannica)

Other very large computer systems in the wide
- Google computers dissipate 300MW on average (150,000 households, $\frac{1}{3}$ reactor)
- Botnets: BredoLab estimated to control 30 millions of zombie computers
- In addition, these systems are heterogeneous and dynamic

Computational Science of Computer Systems

This *essential complexity* mandates adapted scientific instruments

Research Field: Methodologies of Experimentation
- Assessing the performance and correctness of **large-scale computer systems**
- Meta-research on **producing scientifically sound results**
- Main contribution: **SimGrid, a large-scale computer systems simulator**

First title (rejected)

Simulating Applications for **Research** in
Simulation Applications for Research

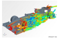
Epistemological Stance
- Empirically consider large-scale computer systems as natural objects
- Eminently artificial artifacts, but complexity reaches "natural" levels
- Other sciences routinely use computers to understand complex systems

Assessing Distributed Applications

Correctness Study ⇝ Formal Methods
- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study ⇝ Experimentation
- Maths: Often not sufficient to fully understand these systems



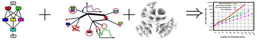


Courtesy of Lucas Nussbaum

- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Emulation: Real applications on Synthetic platforms *(in vitro)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

Simulating Distributed Systems

Big Idea: Simulation is the fastest path from idea to scientific results



Comfort to the user
- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, ignore technical subtleties (for now)

Challenges for the tools
- Validity: Get realistic results (controlled experimental bias)
- Scalability: *Fast enough* and *Big enough*; Tooling: runner, post-processing

Scientific practices sometimes unfortunate in this field
- Experimental settings not detailed enough in literature
- Many short-lived simulators; few sound and established tools

SimGrid: Versatile Simulator of Distributed Apps

Scientific Instrument
- Versatile: Grid, P2P, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Community-driven: 30 contributors (5 not affiliated), 5 contributed tools, GPL

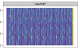
Scientific Object
- Allows comparison of network models on non-trivial applications
- High-Performance Simulation on realistic workload
- Full model checker of distributed applications; Emulator under way

Large Established Project
- Started in 1998; Collab. Loria / Inria Grenoble / CC-IN2P3 / U. Hawaii
- Impact: 120 publications (110 distinct authors, 5 continents), 4 PhD
- Co-leader with A. Legrand (CNRS Grenoble) and F. Suter (CNRS IN2P3)

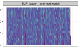
Simulation Validity

SotA: Models in most simulators are either simplistic, wrong or not assessed
- PeerSim: discrete time, application as automaton;
- GridSim/CloudSim: naive packet level or buggy flow sharing
- OptorSim, GroudSim: documented as wrong on heterogeneous platforms

SimGrid provides several Network Models
- Flow-based: Contention, Slow-start, TCP congestion, Cross-traffic effects
- Constant time: A bit faster, but no hope of realism
- Coordinate-based: Easier to instantiate in P2P scenarios
- Packet-level: NS3 bindings



Real Sweep3D



Simulated Sweep3D

Major Contributions (with many contributors)

1/ **Proto-Emulation:** Assessing Real Applications
- GRAS: Middleware to run simulation prototypes on real platforms
- SMPI: Study real MPI applications within SimGrid

2/ **HPS:** High Performance and Scalable Simulation
- Fast Enough: Innovative PDES; Efficient algorithms and implementations
- Big Enough: Scalable and versatile platform representation

3/ **Formal:** Correctness Studies in SimGrid
- Seamless integration of a complete Model Checker (enforces code invariants)
- Exhaustive reachability analysis, with innovative versatile DPOR technique

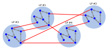
Scientific Community Management
- Project Coordinator: 2 ANR projects, 1 regional CPER project (total: 4M€)
- Methodological convergence: Board member of Grid'5000 experimental grid
- Scientific Animation (SimGrid, Grid'5000): 4 summer schools, 3 R&D engineers

+ leading role in teaching, pedagogical tools, popularization and didactic projects

Parallel Simulation of Discrete Event Systems

- 30 years of literature on efficient Simulation Engines, FES and distribution
- Yet, all DES simulator for P2P were sequential (but dPeerSim)

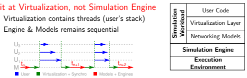
The dPeerSim attempt
- Distributed implementation of PeerSim
- Classical parallelization: spreads the load over several Logical Processes (LP)



Evaluation
- Uses Chord as a standard workload: e.g. 320,000 nodes ⤳ 320,000 requests
- Very good speedup results: 4h on 2 LPs ⤳ 1h on 16 LPs
- But 47s in the original sequential PeerSim (and 5s in precise SimGrid)
- Yet, best known parallelization of DES simulator of P2P systems

New Parallelization Schema for DES

Split at Virtualization, not Simulation Engine
- Virtualization contains threads (user's stack)
- Engine & Models remains sequential



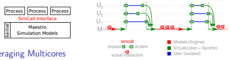
Understanding the trade-off
- Sequential time: $\sum_{SR}(engine + model + virtu + user)$
- Classical schema: $\sum_{SR}\left(\max_{i\in LP}(engine_i + model_i + virtu_i + user_i) + proto\right)$
- Proposed schema: $\sum_{SR}\left(engine + model + \max_{i\in WT}(virtu_i + user_i) + sync\right)$
- Synchronization protocol expensive wrt the engine's load to be distributed

Toward Parallel P2P Simulation in SimGrid

Keep models sequential, execute processes in parallel

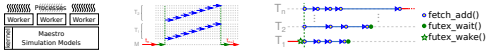
OS-inspired Approach toward Process Separation
- Fine-locking would be difficult, inefficient and would hinder reproducibility
- Mediate any process interactions through simcalls (conceptually identical to *syscalls* of real OSes)

Functional View — Temporal View



Leveraging Multicores

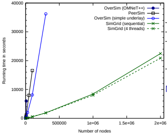
⇒ More processes than cores ⤳ Worker Threads (execute co-routines ;)



Functional View — Temporal View — Ideal Algorithm

Sequential Performance in State of the Art

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



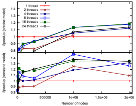
Largest simulated scenario

	Size	Time
Omnet++	10k	1h40
PeerSim	100k	4h36
OverSim	300k	10h
SG, precise	10k	130s
	300k	32mn
	2M	6h23
SG, simple	2M	5h30

Memory Usage
- 2M precise nodes: 32 GiB
- That is 18kiB per process (User stack: 12kiB)

Extra complexity of parallel execution don't impact sequential performance

Benefits of the Parallel Execution



- Speedup $\left(\frac{t_{seq}}{t_{par}}\right)$: up to 45%
- More efficient with simple model:
  - Less work in engine + Amhdal law
- Speedup depends on thread amount
  - 8 threads (of 24 cores) often better
  - Synch costs remain hard to amortize
  - They depend on thread amount

Parallel Efficiency $\left(\frac{speedup}{\#cores}\right)$ for 2M nodes

Model	4 threads	8 th.	16 th.	24 th.
Precise	0.28	0.15	0.07	0.05
Constant	0.33	0.16	0.08	0.06

- Baaaaad efficiency results
- Remember, P2P and Chord: Worst case scenarios

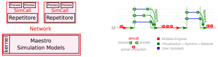
Yet, first time that Chord's parallel simulation is faster than best known sequential

Future Work on HPS

Distributed Simulation toward size
- Leverage the memory of more nodes; Useless in P2P, more adapted to SMPI

Design: split our design under the simcall layer

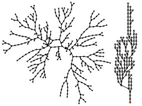
Functional View — Temporal View



Increase level of parallelism
- Pessimistic execution (as now): efficient for 500,000 processes and more...
- Optimistic execution unfeasible because of our complex state
- Vision: realistic execution run optimistically only if it is safe to do so
  Determining independent actions is easy using formal methods

Exhaustive Testing for Correctness Formal Assesment

Model Checking's Big Idea
- Explore all possible executions of the system
- Actively searching for property violations

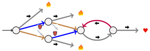


Testing can only prove the presence of bugs. — Dijkstra

well, unless it's exhaustive :)

Model Checking in Wonderland

A warrior seeks her prince.
She can grab ⟍, grab ⛊, move ➡, move ⬅.



Model checking: Actively search for a counter example
- If not found, then the property was true after all
- If found, we got a counter-example (very precious during bug squashing)

Safety Property: $\square(\neg$ 🔥$)$
- Search an invalidating state
- Exhaustive traversal: property true

Liveness Property: $\square((⟍ \wedge ⛊) \Rightarrow \diamond ♥)$
- Search a cycle w/ property is false
- Counter-example is infinite

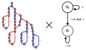
The Problem with Model Checking

I use programs, not models

- Model-checking usually done on logical models, e.g. expressed with TLA$^+$
- Some technics require the full graph, that I never have
- ⇒ Explicit exploration of Implicit graph is called Dynamic Verification

Liveness Properties

- Nice properties are liveness ones, not safeties, but that's much harder
- Counter example must be of infinite length, so encoded as Buchi automaton



Any process that asks the critical section will get it

- r: request
- cs: critical section
- LTL property: $\square(r \Rightarrow \lozenge cs)$

State-space Explosion

- Nice problems require $2^{2^{100}}$ years in practice (or more)
- Several reduction technics exists, but preserving cycles is harder

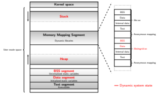
Dynamic Verification in SimGrid

Verifying safety properties

- It works (MSG & SMPI); Reduction with DPOR-based reduction techniques
- Found *wild* bugs in medium-sized programs (Chord protocol)

Verifying liveness properties (ongoing)

- Problem: detect when the system reenters an (accepting) state
- We need system-level state equality



- Byte-per-byte comparison ineffective
- Lots of false negatives (aka undetected violations)

Challenges of System-level State Equality

Over provisioning

Syntactic differences

- In malloc, blocs order can vary without impacting applicative semantic

Padding Bytes

- Data is aligned in memory for efficiency, leaving holes

Irrelevant differences

- Host-related data (pid, files), simulation-related data (time)

Toward Liveness Properties in SimGrid

System Solutions to this Formal Problem

Problem	Heap solution	Stack solution
Over provisioning	Memset 0 + requested size	Stack pointer
Padding bytes	Memset 0	DWARF + libunwind
Irrelevant differences	MC_ignore	DWARF + libunwind
Syntactic differences	Canonicalization	N/A

Current state

- Toy artificial bugs found; Toy property on non-tivial code (NeverJoin in Chord)
- State equality gives a new reduction that works on liveness, too

Future

- MPI3 asynchrone collective operations are a call for semantic bugs
- Assessing properties on communication schema toward easier checkpointing
- Assessing linearizability (service is robust to concurrent usages)
- Explore specific reduction techniques for distributed apps

Take Away Messages

SimGrid will prove helpful to your research

- Versatile: Used in several communities (scheduling, GridRPC, HPC, P2P, Clouds)
- Accurate: Model limits known thanks to validation studies
- Sound: Easy to use, extensible, fast to execute, scalable to death, well tested
- Open: User-community much larger than contributors group; LGPL
- Around since over 10 years, and ready for at least 10 more years

Welcome to the Age of (Sound) Computational Science

SIMGRID

- Discover: `http://simgrid.gforge.inria.fr/`
- Learn: 101 tutorials, user manuals and examples
- Join: user mailing list, #simgrid on irc.debian.org
  We even have some open positions ;)

The Computational Science Nightmare

Computational Science is rarely Reproducible!

- Scientific publications must include all information needed for reproduction
- Knowledge is not the finding, but the method. – Boyle

Issue shared with other scientific disciplines

- *Why Most Published Research Findings are False*. Ioannidis, PloS Med, 2005.
- *Reproducibility in Computational and Experimental Maths* workshop, 12/2012

JASA June	Computational Articles	Code Available
1996	9 of 20	0%
2006	33 of 35	9%
2009	32 of 32	16%
2011	29 of 29	21%

V. Stodden

Non-CS major will teach us about Computational Science!

(inspired from Victoria Stodden, Department of Statistics, Columbia University)

Open Science, and CS^2

Required Tools

- Standard tools: Matlab, R in statistics, ...
- Dissemination Platforms: RunMyCode.org
- Workflow Tracking and Research Environments: VisTrails, MyExperiment.org
- Embedded Publishing: Sweeve
- Journal Policy: Things evolve veeeery slowly

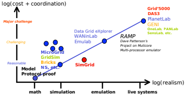
My Research Plan

- SimGrid is a standard tool; use it as troyan to pass best practices along
- Ease experiment packaging and sharing
- Increase associated tools (adaptative runners) to increase the incentive
- Improve our own best practices within the team
- Learn from other disciplines, and build upon this

Conclusion

Scientific Instruments for Distributed Systems

- Common Belief in 2008: Simulation as a toy methodology
- Consensus in 2013: SimGrid as a scientific instrument (w/ Grid'5000)
- Consensus in 2020? We were naïve in 2010, but it works better now



Simulation turned into a reliable scientific instrument!

But there is still a long way to go!

Research Program

Computational Science *of* Computer Systems

pursued convergence of Simulation, Dynamic Verification and Emulation

1/ Modeling of Large-Scale Systems

- Scalability and Accuracy still not enough for Exascale studies
- Semantic modeling of MPI 3.0 collectives (implementation-depend)

2/ Formal Methods for Large-Scale and HPC Systems

- Liveness properties on legacy code (OS-level introspection tooling)
- Domain-specific properties and reduction techniques

3/ Simulation of Real Applications

- OS Virtualization layer for the simulation of legacy code
- Distributed simulation, and increase parallelism in our simulation

4/ Scientific Instrument and Open Science

- Produce a *de facto* standard tool, with associated tools
- Foster the emergence of a vivid research community, with best practices

Question slides

Emulating Large-Scale Applications

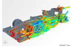
Execute your application in a perfectly controlled environment

- Real platforms are not controllable, so how to achieve this?
- Let's look at what engineers do in other fields

When you want to build a race car...

...adapted to wet tracks

...in a dry country ...



...you can simulate it.

But then, you have

- To assess models
- Technical burden
- No real car

Why don't you...

just control the climate?

or tweak the car's reality?

Courtesy of Lucas Nussbaum

GRAS (Grid Reality And Simulation)

Personal Use: develop real applications within the simulator

Develop Once, Deploy Twice

- Develop and tune on the simulator; Deploy *in situ* without modification
  How: One API, two implementations

Grid Runtime Environment (result = application $\neq$ prototype)

- Performance: efficient wire protocol for structured data
- Portable: across OSes, across CPU architectures, zero dependency

But this forces an API to the users!

Simulated MPI: Simulating real MPI applications

Online simulation of unmodified MPI application within SimGrid

- Algorithm prototyping; Platform dimensionning; What-if analysis ...

PB 1: Enable this mode of MPI execution

- (partially) Reimplement MPI on top of SimGrid
- Fold MPI processes as threads
- Allow to manually factorize data memory

PB 2: Useless if not realistic enough

- Improve model ⇝ piece-wise linear model
  Accurate also for small messages
- Preserve good modeling of network contention

SMPI Future Work

Improve the enabling of MPI simulation

- Passes (almost) all MPICH tests
- Privatization of variable still difficult ⇝ separate MPI processes
- Simulate 10^6 MPI Linpack processes within SimGrid?
- Distribute simulation to achieve this size-up

Push the validity limit further

- Validity is acceptable on simple examples
- Further improve the modeling of one-to-one communications
- Model global communications (OpenMPI vs. MPICH2)
- Model CPU and memory performance (with MESCAL team)

Vision

- Be the best alternative to simulate ExaScale Systems
- ANR SONGS project coordinates these efforts (tool versatility considered helpful)

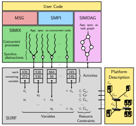
Quick Overview of Internals Organization

User-visible SimGrid Components

- MSG: heuristics as Concurrent Sequential Processes (Java/Ruby/Lua bindings)
- SimDag: heuristics as DAG of (parallel) tasks
- SMPI: simulate real applications written using MPI

SimGrid internal layers

- MSG: User-friendly syntactic sugar
- Simix: Processes, synchronizations
- SURF: Resources usage interface
- Models: Compute completion dates



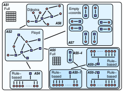
SimGrid Scalability

Simulation Versatility should not hinder Scalability

- Two aspects: Big enough (large platforms) ⊕ Fast enough (large workload)

Versatile yet Scalable Platform Descriptions

- Hierarchical organization in ASes
  ⇝ cuts down complexity
  ⇝ recursive routing
- Efficient on each classical structures
  Flat, Floyd, Star, Coordinate-based
- Allow bypass at any level

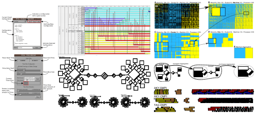


⇝ Grid'5000 platform in 22KiB
(10 sites, 40 clusters, 1500 nodes)

⇝ King's dataset in 290KiB
(2500 nodes, coordinate-based)

Visualizing SimGrid Simulations

- Visualization scriptable: easy but powerful configuration; Scalable tools
- Right Information: both platform and applicative visualizations
- Right Representation: gantt charts, spatial representations, tree-graphs
- Easy navigation in space and time: selection, aggregation, animation
- Easy trace comparison: Trace diffing (still partial ATM)

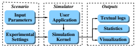


Other Associated Tools

Workflow to any Experiments through Simulation

1. Prepare the experimental scenarios
2. Launch thousands of simulations
3. Post-processing and result analysis

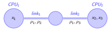
⇝ Each simulation is only a brick



Workload Generation

- Platforms: Simulacrum (generation), PDA (archive) and MintCAR (mapping)
- Applicative Workload: Tau-based trace collection + replay

Max-Min Fairness between Network Flows



$$x_1 \leq Power\_CPU_1 \quad (1a)$$
$$x_2 + x_3 \leq Power\_CPU_2 \quad (1b)$$
$$\rho_1 + \rho_2 \leq Power\_link_1 \quad (1c)$$
$$\rho_1 + \rho_3 \leq Power\_link_2 \quad (1d)$$

Computing the sharing between flows

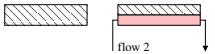
- Objective function: maximize $\min_{f \in \mathcal{F}}(\rho_f)$ [Massoulié & Roberts 2003]
- Equilibrium: increasing any ρ_f decreases a ρ'_f (with $\rho_f > \rho'_f$)
- (actually, that's a simplification of our real objective function)

Efficient Algorithm

1. Search for the bottleneck link l so that: $\frac{C_l}{n_l} = min\left\{\frac{C_k}{n_k}, k \in \mathcal{L}\right\}$
2. This determines any flow f on this link: $\rho_f = \frac{C_l}{n_l}$
3. Update all n_l and C_l to remove these flows; Loop until all ρ_f are fixed

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = 0$ $\quad n_1 = 0$
$C_2 = 0$ $\quad n_2 = 0$

$\rho_0 = C/2$
$\rho_1 = C/2$
$\rho_2 = C/2$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1
- ⇒ $\rho_0 = C/2$ and $\rho_1 = C/2$
- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

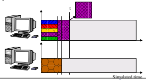
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s ⊕ Computation = amount A of Flops ⇝ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



How big and how fast? (1/3 – Grid and VC)

Comparison to GridSim

A master distributes 500,000 fixed size jobs to 2,000 workers (round robin)

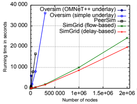
	GridSim	SimGrid
Network model	delay-based model	flow model
Topology	none	Grid5000
Time	1h	14s
Memory	4.4GB	165MB

Volunteer Computing settings

- Loosely coupled scenario as in Boinc
- SimGrid: full modeling (clients and servers), precise network model
- SimBA: Servers only, descisions based on simplistic markov modeling
- ⇝ SimGrid shown 25 times faster

How big and how fast? (2/3 – P2P)

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



Largest simulated scenario

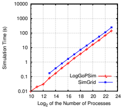
Simulator	size	time
OverSim (OMNeT++)	10k	1h40
OverSim (simple)	300k	10h
PeerSim	100k	4h36
SG (flow-based)	10k	130s
	300k	32mn
	2M*	6h23
SG (delay-based)	2M	5h30

* 36GB = 18kB/ process (16kB for the stack)

- Orders of magnitude more scalable than state-of-the-art P2P simulators
- Precise model incurs a ≈ 20% slowdown, but accuracy is not comparable
- Also, parallel simulation (faster simulation at scale); Distributed sim. ongoing

How big and how fast? (3/3 – HPC)

Simulating a binomial broadcast



Model:

- SimGrid: contention + cabinets hierarchy
- LogGOPSim: simple delay-based model

Results:

- SimGrid is roughly 75% slower
- SimGrid is about 20% more fat (15GB required for 2^{23} processors)

The genericity of SimGrid data structures comes at the cost of a slight overhead
BUT scalability does not necessarily comes at the price of realism

Contributions to Experimental Facilities (in vivo)

Grid'5000 Project: world leading scientific instrument for dist. apps

- Instrument for research in computer science (*deployment* of customized OSes)
  1500 nodes (2800 cpus, 7200 cores). 9 sites: dedicated 10Gb network

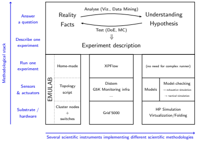
Personal Contributions

- National steering committee; Local project co-leader (CPER, Aladdin, Hemera)
- Scientific animation, event co-organization: Nancy is a leading site
- Collaboration: Production grids (IdG), CEA, Arcelor-Mittal

Project: Experimentation Process Industrialization (with L. Nussbaum)

- Open science: ensure that experiments can be shared, reviewed, improved
- Convergence of simulation and direct execution
- Methodological framework and practical tools (+administrative duties)

One Methodology to Rule Them All

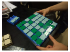


Médiation scientifique

Sciences Manuelles du Numérique

- Faire des activités d'initiation à la science informatique
- Pour la fête de la science, pour les TS (1/3 du temps hors machine)

- Boolier: Codage binaire de l'information, code correcteur, transmission
- Crêpier psycho-rigide: Notion d'algorithme, tri
- Base-ball coloré: Algorithme, algorithme efficace, algorithme correct
- Robozzle: Programmation (instruction, boucle, fonction)



Emulating Large-Scale Applications

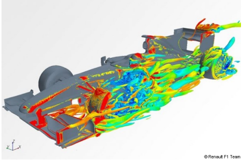
Execute your application in a perfectly controlled environment

- Real platforms are not controllable, so how to achieve this?
- Let's look at what engineers do in other fields

When you want to build a race car. . .

. . . adapted to wet tracks

. . . in a dry country . . .



. . . you can simulate it.

But then, you have

- To assess models
- Technical burden
- No real car

Why don't you. . .

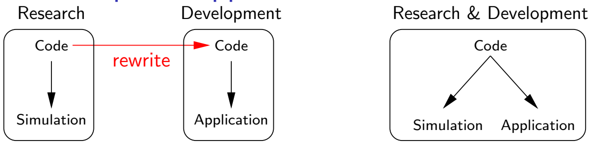
just control the climate?

or tweak the car's reality?

Courtesy of Lucas Nussbaum

GRAS (Grid Reality And Simulation)

Personal Use: develop real applications within the simulator



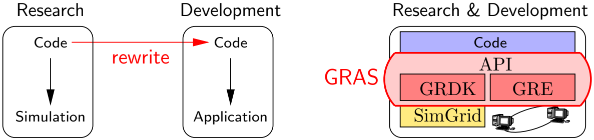
Without GRAS   With GRAS

Develop Once, Deploy Twice

- Develop and tune on the simulator; Deploy *in situ* without modification

GRAS (Grid Reality And Simulation)

Personal Use: develop real applications within the simulator



Without GRAS   With GRAS

Develop Once, Deploy Twice

- Develop and tune on the simulator; Deploy *in situ* without modification

  How: One API, two implementations

Grid Runtime Environment (result = application $\neq$ prototype)

- Performance: efficient wire protocol for structured data
- Portable: across OSes, across CPU architectures, zero dependency

But this forces an API to the users!

Simulated MPI: Simulating real MPI applications

Online simulation of unmodified MPI application within SimGrid

- Algorithm prototyping; Platform dimensionning; What-if analysis ...

PB 1: Enable this mode of MPI execution

- (partially) Reimplement MPI on top of SimGrid
- Fold MPI processes as threads
- Allow to manually factorize data memory

PB 2: Useless if not realistic enough

- Improve model $\rightsquigarrow$ piece-wise linear model
  Accurate also for small messages
- Preserve good modeling of network contention

SMPI Future Work

Improve the enabling of MPI simulation

- Passes (almost) all MPICH tests
- Privatization of variable still difficult $\rightsquigarrow$ separate MPI processes
- Simulate 10^6 MPI Linpack processes within SimGrid?
- Distribute simulation to achieve this size-up

Push the validity limit further

- Validity is acceptable on simple examples
- Further improve the modeling of one-to-one communications
- Model global communications (OpenMPI vs. MPICH2)
- Model CPU and memory performance (with MESCAL team)

Vision

- Be the best alternative to simulate ExaScale Systems
- ANR SONGS project coordinates these efforts (tool versatility considered helpful)

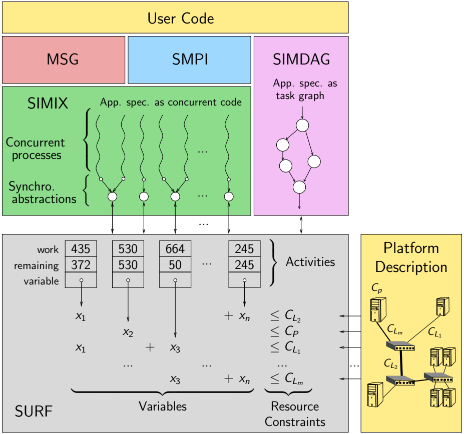
Quick Overview of Internals Organization

User-visible SimGrid Components

- MSG: heuristics as Concurrent Sequential Processes (Java/Ruby/Lua bindings)
- SimDag: heuristics as DAG of (parallel) tasks
- SMPI: simulate real applications written using MPI

SimGrid internal layers

- MSG: User-friendly syntactic sugar
- Simix: Processes, synchronizations
- SURF: Resources usage interface
- Models: Compute completion dates



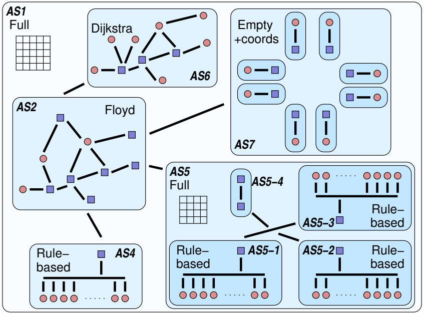
SimGrid Scalability

Simulation Versatility should not hinder Scalability

- Two aspects: Big enough (large platforms) $\oplus$ Fast enough (large workload)

Versatile yet Scalable Platform Descriptions

- Hierarchical organization in ASes
  $\leadsto$ cuts down complexity
  $\leadsto$ recursive routing
- Efficient on each classical structures
  Flat, Floyd, Star, Coordinate-based
- Allow bypass at any level

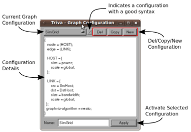


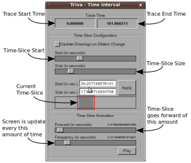
$\leadsto$ Grid'5000 platform in 22KiB
(10 sites, 40 clusters, 1500 nodes)

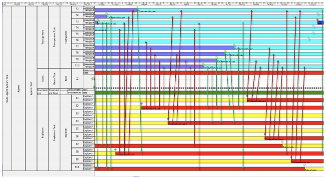
$\leadsto$ King's dataset in 290KiB
(2500 nodes, coordinate-based)

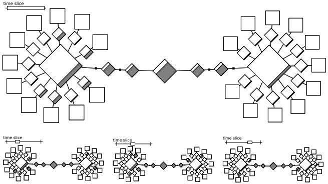
Visualizing SimGrid Simulations

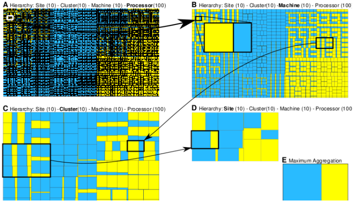
- Visualization scriptable: easy but powerful configuration; Scalable tools
- Right Information: both platform and applicative visualizations
- Right Representation: gantt charts, spatial representations, tree-graphs
- Easy navigation in space and time: selection, aggregation, animation
- Easy trace comparison: Trace diffing (still partial ATM)

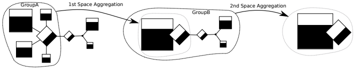


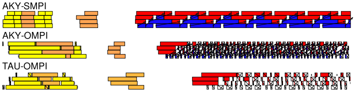










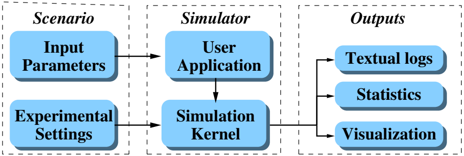


Other Associated Tools

Workflow to any Experiments through Simulation

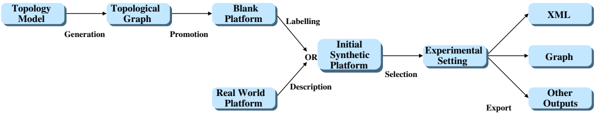
1. Prepare the experimental scenarios
2. Launch thousands of simulations
3. Post-processing and result analysis

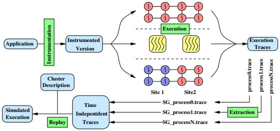
⇝ Each simulation is only a brick



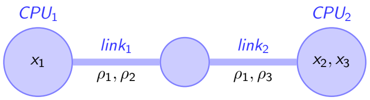
Workload Generation

- Platforms: Simulacrum (generation), PDA (archive) and MintCAR (mapping)
- Applicative Workload: Tau-based trace collection + replay





Max-Min Fairness between Network Flows



$$x_1 \leq Power\_CPU_1 \quad (1a)$$
$$x_2 + x_3 \leq Power\_CPU_2 \quad (1b)$$
$$\rho_1 + \rho_2 \leq Power\_link_1 \quad (1c)$$
$$\rho_1 + \rho_3 \leq Power\_link_2 \quad (1d)$$

Computing the sharing between flows

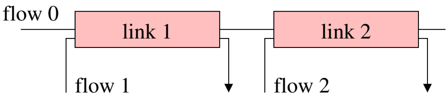
- Objective function: maximize $\min_{f \in \mathcal{F}}(\rho_f)$ [Massoulié & Roberts 2003]
- Equilibrium: increasing any ρ_f decreases a ρ'_f (with $\rho_f > \rho'_f$)
- (actually, that's a simplification of our real objective function)

Efficient Algorithm

1. Search for the bottleneck link l so that: $\frac{C_l}{n_l} = min\left\{\frac{C_k}{n_k},\ k \in \mathcal{L}\right\}$
2. This determines any flow f on this link: $\rho_f = \frac{C_l}{n_l}$
3. Update all n_l and C_l to remove these flows; Loop until all ρ_f are fixed

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = C \qquad n_1 = 2$

$C_2 = C \qquad n_2 = 2$

$\rho_0 =$

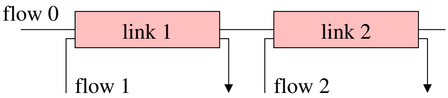
$\rho_1 =$

$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = C \qquad n_1 = 2$

$C_2 = C \qquad n_2 = 2$

$\rho_0 = C/2$

$\rho_1 = C/2$

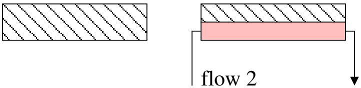
$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = 0 \qquad n_1 = 0$
$C_2 = C/2 \qquad n_2 = 1$

$\rho_0 = C/2$
$\rho_1 = C/2$
$\rho_2 =$

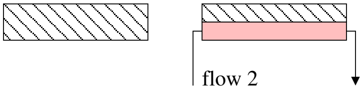
- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = 0 \quad n_1 = 0$
$C_2 = 0 \quad n_2 = 0$

$\rho_0 = C/2$
$\rho_1 = C/2$
$\rho_2 = C/2$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

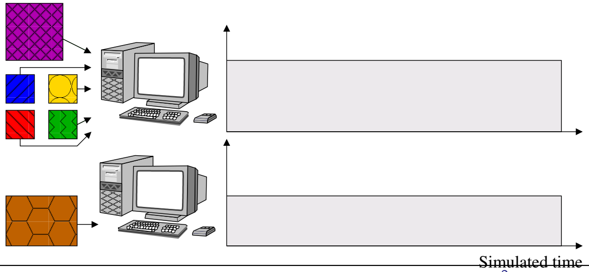
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources



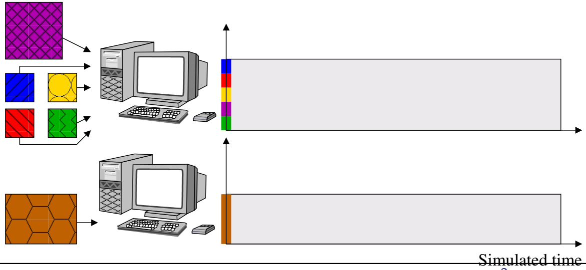
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)



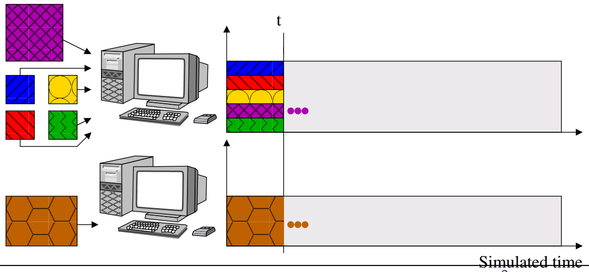
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time



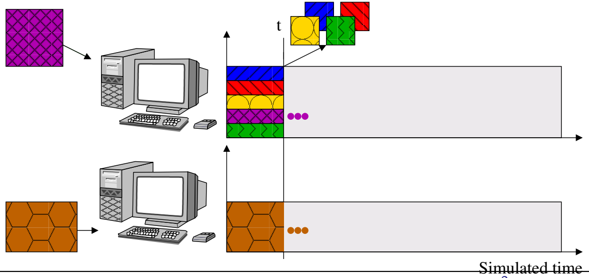
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions



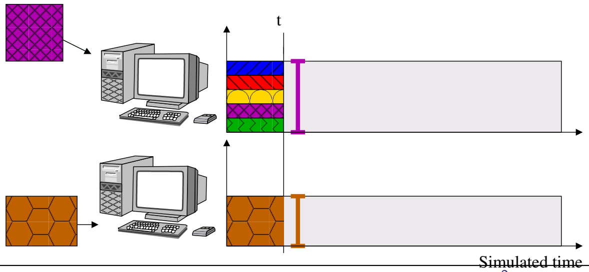
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



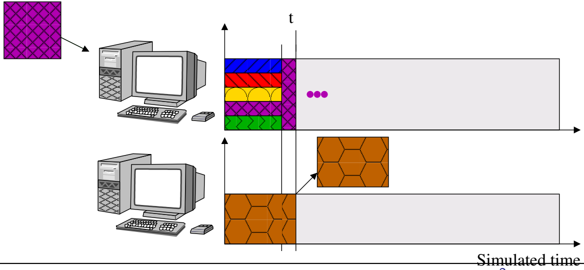
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



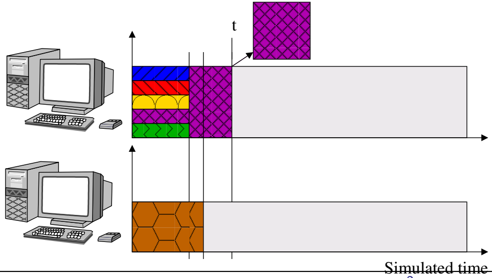
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



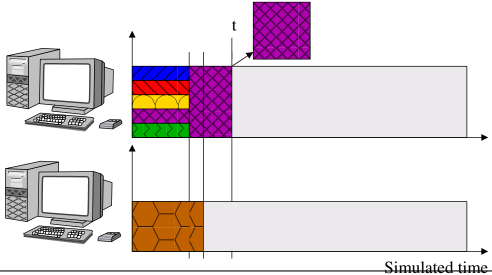
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

How big and how fast? (1/3 – Grid and VC)

Comparison to GridSim

A master distributes 500,000 fixed size jobs to 2,000 workers (round robin)

	GridSim	SimGrid
Network model	delay-based model	flow model
Topology	none	Grid5000
Time	1h	14s
Memory	4.4GB	165MB

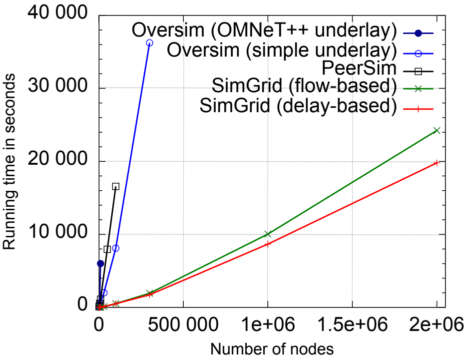
Volunteer Computing settings

- Loosely coupled scenario as in Boinc
- SimGrid: full modeling (clients and servers), precise network model
- SimBA: Servers only, descisions based on simplistic markov modeling

⇝ SimGrid shown 25 times faster

How big and how fast? (2/3 – P2P)

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



Largest simulated scenario

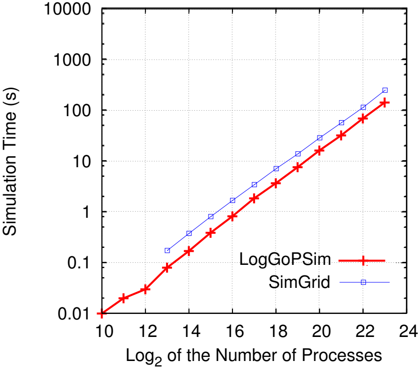
Simulator	size	time
OverSim (OMNeT++)	10k	1h40
OverSim (simple)	300k	10h
PeerSim	100k	4h36
SG (flow-based)	10k	130s
	300k	32mn
	2M*	6h23
SG (delay-based)	2M	5h30

* 36GB = 18kB/ process (16kB for the stack)

- Orders of magnitude more scalable than state-of-the-art P2P simulators
- Precise model incurs a ≈ 20% slowdown, but accuracy is not comparable
- Also, parallel simulation (faster simulation at scale); Distributed sim. ongoing

How big and how fast? (3/3 – HPC)

Simulating a binomial broadcast



Model:

- SimGrid: contention + cabinets hierarchy
- LogGOPSim: simple delay-based model

Results:

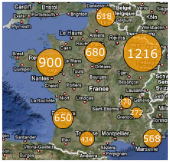
- SimGrid is roughly 75% slower
- SimGrid is about 20% more fat (15GB required for 2^{23} processors)

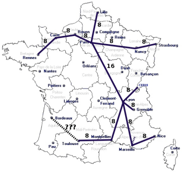
The genericity of SimGrid data structures comes at the cost of a slight overhead
BUT scalability does not necessarily comes at the price of realism

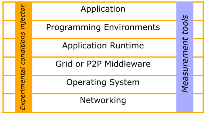
Contributions to Experimental Facilities (in vivo)

Grid'5000 Project: world leading scientific instrument for dist. apps

- Instrument for research in computer science (*deployment* of customized OSes)
  1500 nodes (2800 cpus, 7200 cores), 9 sites; dedicated 10Gb network







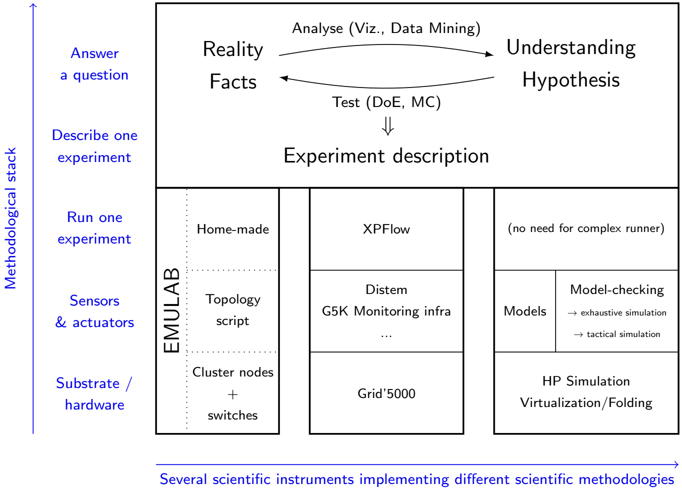
Personal Contributions

- National steering committee; Local project co-leader (CPER, Aladdin, Hemera)
- Scientific animation, event co-organization: Nancy is a leading site
- Collaboration: Production grids (IdG), CEA, Arcelor-Mittal

Project: Experimentation Process Industrialization (with L. Nussbaum)

- Open science: ensure that experiments can be shared, reviewed, improved
- Convergence of simulation and direct execution
- Methodological framework and practical tools (+administrative duties)

One Methodology to Rule Them All



Médiation scientifique

Sciences Manuelles du Numérique

- Faire des activités d'initiation à la science informatique
- Pour la fête de la science, pour les TS (1/3 du temps hors machine)

- Boolier: Codage binaire de l'information, code correcteur, transmission
- Crêpier psycho-rigide: Notion d'algorithme, tri
- Base-ball coloré: Algorithme, algorithme efficace, algorithme correct
- Robozzle: Programmation (instruction, boucle, fonction)