Computational Science of Distributed Systems

Martin Quinson (Université de Lorraine)

November 28, 2012

Initial title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Martin Quinson (Université de Lorraine)

November 28, 2012

Initial title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

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

Martin Quinson (Université de Lorraine)

November 28, 2012

Initial title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Simulation Applications for Research

- Simulation is the third pillar of science (with theory and experiment)
- Computational Science = many simulations + big data
- Grids and HPC: parameter sweeps and simulations by scientists

Initial title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Simulation Applications for Research

- Simulation is the third pillar of science (with theory and experiment)
- Computational Science = many simulations + big data
- Grids and HPC: parameter sweeps and simulations by scientists

Simulating Application

- Assessing CS ideas through real experiments: long, difficult, bothersome
- Simulation makes it easy (but sometimes unsound)
- SimGrid is Versatile, Sound and Open

Initial title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Simulation Applications for Research

- Simulation is the third pillar of science (with theory and experiment)
- Computational Science = many simulations + big data
- Grids and HPC: parameter sweeps and simulations by scientists

Simulating Application

- Assessing CS ideas through real experiments: long, difficult, bothersome
- Simulation makes it easy (but sometimes unsound)
- SimGrid is Versatile, Sound and Open

Computational Science of Distributed Systems

- Large-Scale Infrastructures complexity $\rightsquigarrow$ Scientific assessment
- All available methodologies must be combined

Research Context

Scientific Objects

Large Scale Distributed Systems

- Scientific Computing
- High Performance Computing
- Grids
- Peer-to-peer Systems
- Volunteer Computing
- Cloud Computing

Scientific Questions

Research Context

Large Scale Distributed Systems

Scientific Objects

- Scientific Computing
- High Performance Computing
- Grids
- Peer-to-peer Systems
- Volunteer Computing
- Cloud Computing

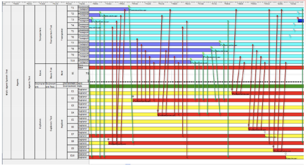
Scientific Questions

Performance

- Time/Energy
- User/Provider
- Throughput/Makespan/#Msg
- Worst case/Avg/Amortized

Correction

- Safety: bad things don't happen
- Liveness: good things do happen


Assessing Distributed Systems

Correction Study $\leadsto$ Formal Methods

- Tests: Unable to provide definitive answers

Performance Study $\leadsto$ Experimentation

- Maths: Often not sufficient to fully understand these systems

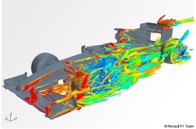
Assessing Distributed Systems

Correction 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



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

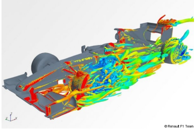
Assessing Distributed Systems

Correction 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



- 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)*

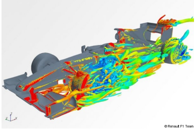
Assessing Distributed Systems

Correction 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



- 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)*

Research Interests: Experimental Methodologies

- Meta-research about how to produce scientifically sound research
- Strive at developing ready-to-use tools addressing methodological challenges

Simulation? Theory is enough for Artificial Artifacts!

Computers contain only what we've put in!

Modern computer systems present an unpreceded complexity

- Heterogeneous components, Dynamic and Complex platforms
- Numerous: milions of cores expected within the decade (ExaScale)
- Large: kernel+jvm+tomcat $\rightsquigarrow$ 50M lines (25 times Encyclopedia Britanica)

Toward a Computational Science of Distributed Computer Systems

- Empirically consider Distributed Systems as "Natural" Objects
- Other sciences routinely use computers to understand complex systems

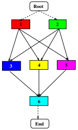
Claim: simulation is both sound and convenient

- Less simplistic than proposed theoretical models
- Easier and faster than experimental platforms
- It should be part of your methodology

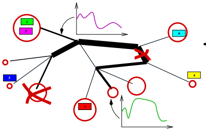
Simulating Distributed Systems

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

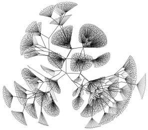
Idea to test | Experimental setup | Simulation Model | Scientific results



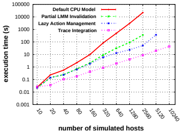
+



+



⇒



Comfort to the user

- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, don't fiddle with technical subtleties (yet)

Challenges for the tools

- Validity: Get realistic results (controlled experimental bias)
- Scalability: Simulate *fast enough* problems *big enough*
- Associated tools: campaign mgmt, result analysis, settings generation, ...
- Applicability: If it doesn't simulate what is important to the user, it's void

Computational Science of Distributed Systems?

Requirements for a Scientific Approach

- Reproducible results: read a paper, reproduce the results and improve
- Standard tools that Grad students can learn quickly

Current practice in the field is quite different

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

	Domain	CPU	Disk	Network	Application	Scale
OptorSim	(Data)Grid	Analytic	Amount.	(buggy) Analytic	Programmatic	1,000
GridSim CloudSim	Grid Cloud	Analytic	Analytic	(buggy) wormhole (buggy) Analytic	Programmatic	1,000
OverSim	P2P	None	None	Euclidian or Pkt-lvl	Programmatic	100,000
PeerSim	P2P	None	None	Constant time	State machine	1,000,000
SimGrid	Grid, VC, P2P, HPC, cloud, ...	Analytic	Amount	Flow, Cste-time or Packet-level (NS3)	Program, Trace or Emulation	1,000,000

SimGrid Framework

Scientific Instrument

- Versatile: Grid, P2P, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Open: Grounded +100 papers; 100 members on simgrid-user@; LGPL

Scientific Object (and lab)

- Workbench for Network Models; Model-Checker; soon Emulator

Scientific Project since 12 years

- Collaboration Loria / Inria Rhône-Alpes / CC-IN2P3 / U. Hawaii
- Fundings INRIA; ANR: USS SimGrid (08-11), SONGS (12-16)

Coming next: SimGrid as a Reliable Scientific Instrument

- High-Performance Simulation for Computer Science
- Formal analysis and Dynamic verification of real applications
- Unified experimental workbench of real applications

SimGrid Scalability (Grids and Volunter Computing)

Simulation Versatility should not hinder Scalability

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

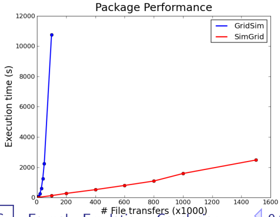
How Big?

P2P	2,500 peers with Vivaldi coordinates	294KB
VC	5120 volunteers	435KB + 90MB
Grid	Grid5000: 10 sites, 40 clusters, 1500 nodes	22KB
HPC	1 cluster of 262144 nodes	5KB
HPC	Hierarchy of 4096 clusters of 64 nodes	27MB
Cloud	3 small data centers + Vivaldi	10KB

How Fast?

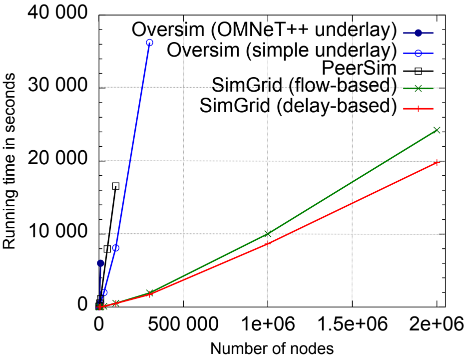
Round robin of 500,000 jobs to 2,000 workers

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



SimGrid Scalability (Peer-to-Peer)

- 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)

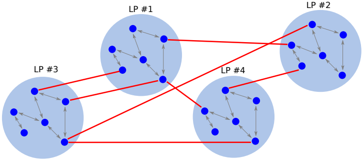
- Orders of magnitude more scalable than state-of-the-art P2P simulators
- Precise model incurs a ≈ 20% slowdown, but accuracy is not comparable
- Next: Can parallel simulation be faster?

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

- Parallel implementation of PeerSim/DES (not by PeerSim main authors)
- 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: 4h10 on 2 LPs, only 1h06 using 16 LPs
- But 47s in the original sequential PeerSim (and 5s in precise SimGrid)
- Yet, best previously known parallelization of DES simulator of P2P systems

New Parallelization Schema for DES

Classical Understanding of Parallel DES

Simulation Workload	▶ Granularity, Communication Pattern ▶ Events population, probability & delay ▶ #simulation objects, #processors
Simulation Engine	▶ Parallel protocol, if any: – Conservative (lookahead, ...) – Optimistic (state save & restore, ...) ▶ Event list mgnt, Timing model...
Execution Environment	▶ OS, Programming Language (C, Java...), Networking Interface (MPI, ...) ▶ Hardware aspects (CPU, mem., net)

Our models are hard to parallelize

- Full linear programs instead of static queues; Evt completion date changes
- They are overly optimized (Cache oblivious, lazy updates)

New Parallelization Schema for DES

Classical Understanding of Parallel DES

Simulation Workload	▶ Granularity, Communication Pattern ▶ Events population, probability & delay ▶ #simulation objects, #processors
Simulation Engine	▶ Parallel protocol, if any: – Conservative (lookahead, . . .) – Optimistic (state save & restore, . . .) ▶ Event list mgnt, Timing model. . .
Execution Environment	▶ OS, Programming Language (C, Java. . .), Networking Interface (MPI, . . .) ▶ Hardware aspects (CPU, mem., net)

Our way of life

<table>
<tr><td rowspan="3">Simulation Workload</td><td>User Code</td></tr>
<tr><td>Virtualization Layer</td></tr>
<tr><td>Networking Models</td></tr>
<tr><td colspan="2">Simulation Engine</td></tr>
<tr><td colspan="2">Execution Environment</td></tr>
</table>

Our models are hard to parallelize

- Full linear programs instead of static queues; Evt completion date changes
- They are overly optimized (Cache oblivious, lazy updates)

That's not the problem anyway

- Performance killer is simulated application itself, not event handling

Toward Parallel P2P Simulation in SimGrid

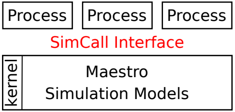
Overall Goal

- Parallelization for speed. Multithreaded on shared memory
- P2P = worst case (fine grain $\rightsquigarrow$ cannot hide issues with app-level parallelism)
- Actually, P2P may not need this but if we succeed here, it works everywhere

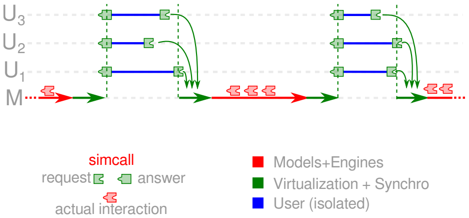
OS-inspired Approach

- Keep models sequential, parallelize the workload: execute processes in parallel
- Processes separation through a OS-oriented approach: simcalls

Functional View



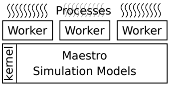
Temporal View



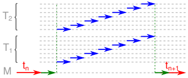
Efficient Parallel Simulation

Leveraging Multicores

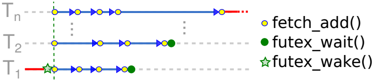
- More processes than cores $\rightsquigarrow$ Worker Threads (execute co-routines ;)



Functional View



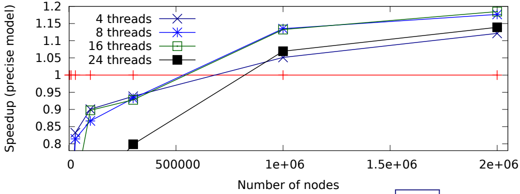
Temporal View



Ideal Algorithm

Reducing Synchronization Costs

- syscalls toward synchronization are the performance killer to optimize
- Assembly reimplementation of ucontext: no syscall on context switch
- Synchronize only at scheduling round boundaries using futexes
- Dynamic load distribution: hardware fetch-and-add next process' index



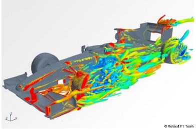
Assessing Distributed Systems

Correction 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



- 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)*

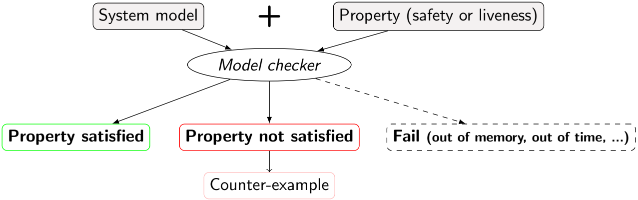
Research Interests: Experimental Methodologies

- Meta-research about how to produce scientifically sound research
- Strive at developing ready-to-use tools addressing methodological challenges

Formal Algorithm Verification

Model-Checking

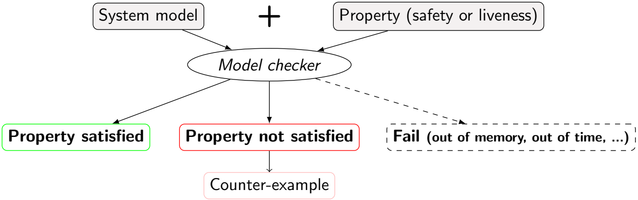
- Automatically checks whether a given model of a system satisfies a property
- Gives a counter-example in case of violation of the property



Formal Algorithm Verification

Model-Checking

- Automatically checks whether a given model of a system satisfies a property
- Gives a counter-example in case of violation of the property

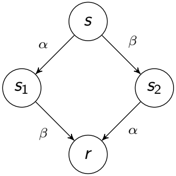


Safety property

- *"Bad things do not append during the execution"*
- **Assertion** on reachabled states

Liveness property

- *"Good things will eventually happen in all cases"*
- Verification on an **execution path**
- **Temporal logic formula** (LTL, CTL, ...)



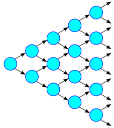
The Problem with Model-Checking

I use programs, not models

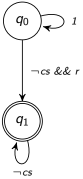
- Model-checking usually done on logical models, e.g. expressed with TLA$^+$
- Some technics require the full graph, that I never have

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



$\times$



- r: request
- cs: critical section
- LTL property: $\Box(r \Rightarrow \Diamond cs)$
  "Any process that asks the critical section must obtain it"

State-space Explosion

- Nice problems are not feasible in practice in less than $2^{2^{100}}$ years
- Several reduction technics exists, but often not for liveness properties

Dynamic Verification in SimGrid

Current state

- Can verify safeties on unmodified programs (model explored implicitely)
- DPOR-based reduction technique integrated
- Found *wild* bugs in medium-sized programs (Chord protocol)

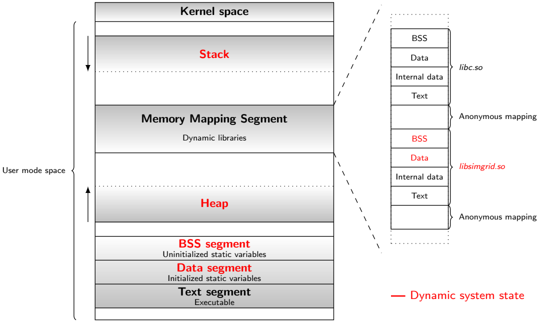
Dynamic Verification in SimGrid

Current state

- Can verify safeties on unmodified programs (model explored implicitely)
- DPOR-based reduction technique integrated
- Found *wild* bugs in medium-sized programs (Chord protocol)

Ongoing work: toward liveness properties

- Problem: detect when the system reenters an (accepting) state

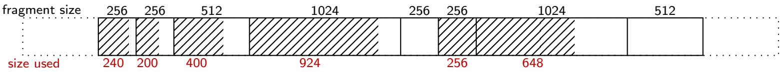


Challenges

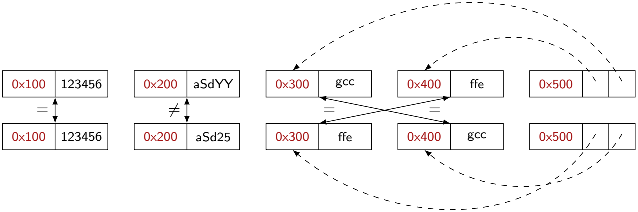
- Memory overprovisionning
- Padding bytes
- Irrelevant OS differences
- Syntaxic differences

Challenges of System-level State Equality

- Overprovisionning



- Padding bytes
- Irrelevant differences about simulation
- Syntactic differences



Toward Liveness Properties in SimGrid

System Solutions to this Formal Problem

Problem	**Heap solution**	**Stack solution**
Overprovisionning	Memset 0 + requested size	Stack pointer
Padding bytes	Memset 0	DWARF + Libunwind
Irrelevant OS differences	MC_ignore	DWARF + libunwind + MC_ignore
Syntactic differences	Canonicalization	N/A

Preliminary results

- Toy artificial bugs found; Toy property on non-tivial code (NeverJoin in Chord)
- State equality gives a new reduction that works on liveness, too
- Difficulty: we are also model-checking SimGrid; hidden bugs strike back

Future

- MPI3 asynchrone collective operations are a call for semantic bugs
- Assessing properties on communication schema toward easier checkpointing

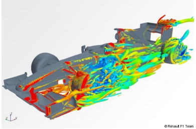
Assessing Distributed Systems

Correction 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



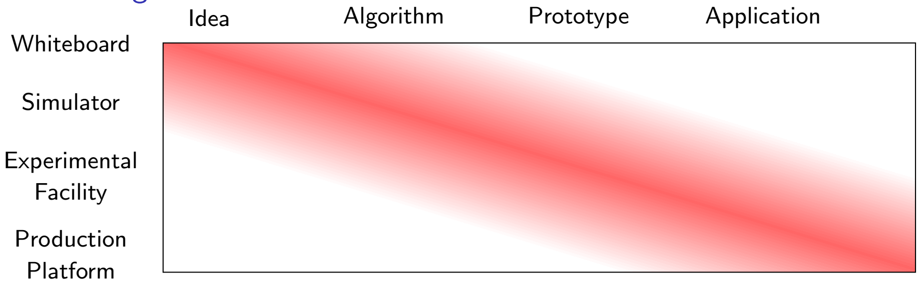
- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Emulation: Real applications on Synthetic platforms *(in vitro)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

Research Interests: Experimental Methodologies

- Meta-research about how to produce scientifically sound research
- Strive at developing ready-to-use tools addressing methodological challenges

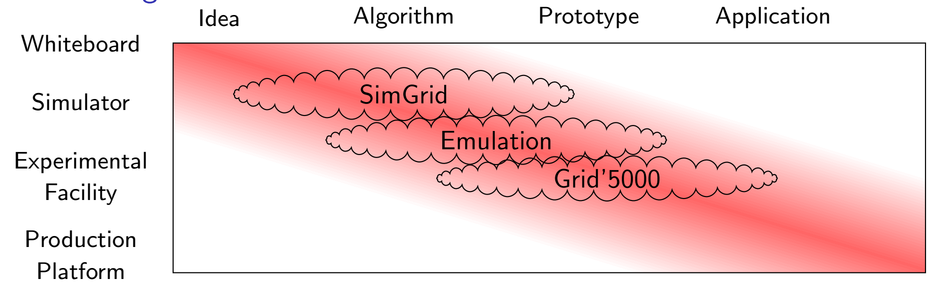
No Experimental Methodology is Sufficient

Methodologies must be Combined



No Experimental Methodology is Sufficient

Methodologies must be Combined



One Workbench to Rule Them All

- Share XP description, DoE and visualization tools
- Dream: seamlessly switch to the most adapted tool
- Ambitious goal, but science is a team game, isn't it?

Coming next: bridging the gap between simulation and real world

Emulation as an Experimental Methodology

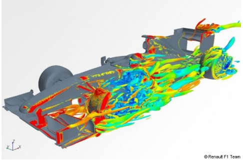
Execute real application in a perfectly controlled environment

- 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.

Courtesy of Lucas Nussbaum

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?

Simulated MPI: Simulating real applications

Online simulation of unmodified MPI application within SimGrid

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

Simulated MPI: Simulating real 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 toy examples
- 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)

How to Emulate Any Application?

Limits of existing approaches

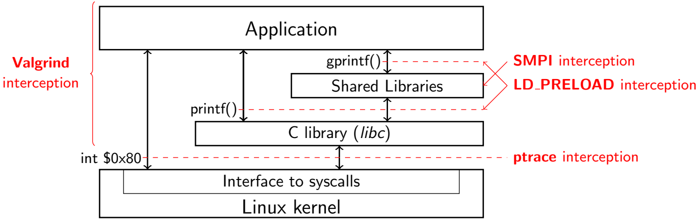
- SMPI is obviously limited to MPI applications (J2EE?)
- Emulation through degradation only reduces the host platform

How to Emulate Any Application?

Limits of existing approaches

- SMPI is obviously limited to MPI applications (J2EE?)
- Emulation through degradation only reduces the host platform

Possible Approaches



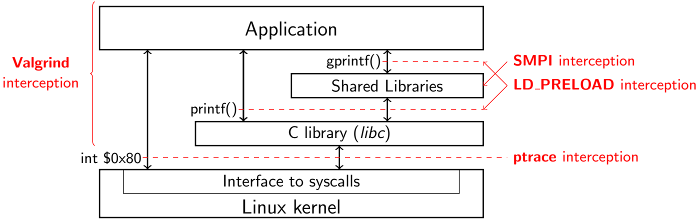
- SMPI: Source-to-source rewrite; • Valgrind: Binary rewrite (slow!)
- LD_PRELOAD: Dynamic loader tricks; • ptrace: syscall trapping

How to Emulate Any Application?

Limits of existing approaches

- SMPI is obviously limited to MPI applications (J2EE?)
- Emulation through degradation only reduces the host platform

Possible Approaches



- SMPI: Source-to-source rewrite; • Valgrind: Binary rewrite (slow!)
- LD_PRELOAD: Dynamic loader tricks; • ptrace: syscall trapping

Current State of simterpose

- Working POC on top of SimGrid, but student code quality for now

Take Away Messages

SimGrid will prove helpful to your research

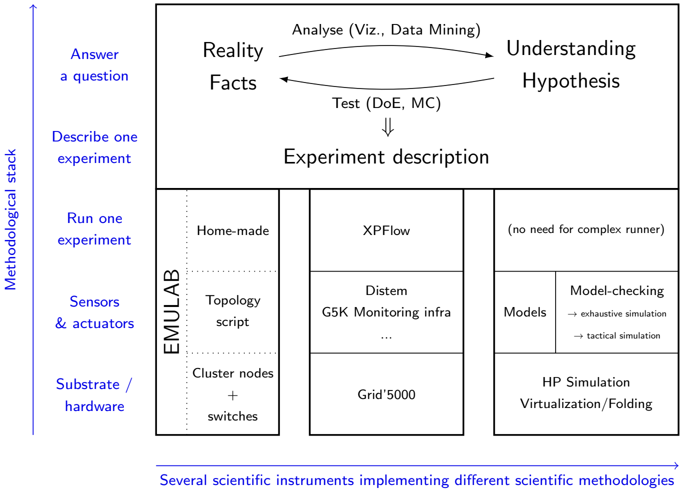
- Versatile: Used in several communities (scheduling, GridRPC, HPC, P2P, Clouds)
- Accurate: Model limits known thanks to validation studies
- Sound: Easy to use, extensible, fast to execute, scalable to death, well tested
- Open: User-community much larger than contributors group; LGPL
- Around since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

One Methodology to Rule Them All



Conclusions

- There is no alternative to Computational Science of Distributed Systems
- Science is Team Game: I have elements, but need a (full) team support