Computational Science of Computer Systems

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

Martin Quinson

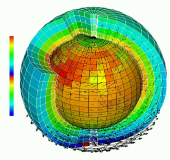
(with the SimGrid Team and others)

October 16th 2014
Mons

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

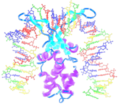
- Last few centuries
- Equations-based
- Can understand
- Prediction long

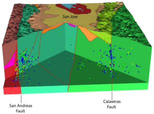
Computational Science

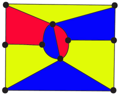
- Nowadays
- Compute-intensive
- Can simulate
- Prediction easier

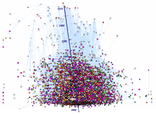
Prediction is very difficult, especially about the future. – Niels Bohr

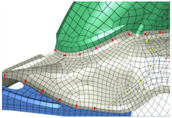
Computational Science

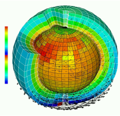




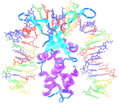


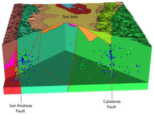


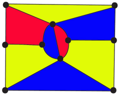


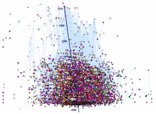


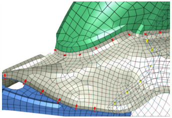
Computational Science

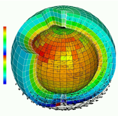




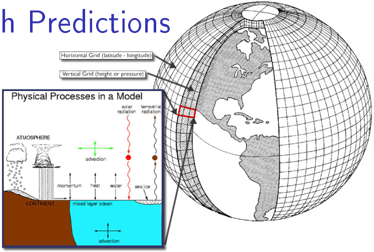




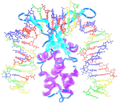
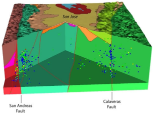


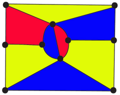
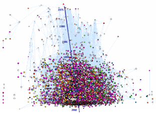
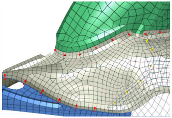
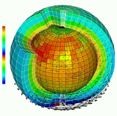


Understanding the Climate Change with Predictions



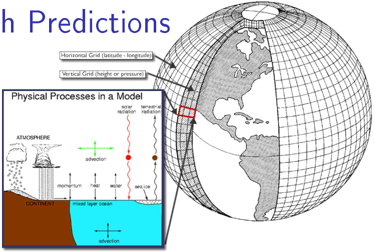
Computational Science

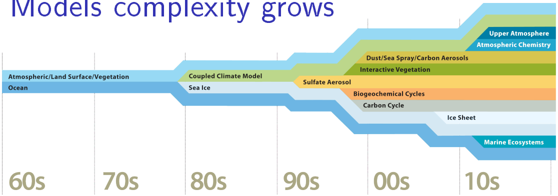





Understanding the Climate Change with Predictions





Models complexity grows



this requires
large computers

Modern Computers are Large and Complex

Massive Parallelism

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

ExaScale Systems, used in Computational Science

- Systems doing one Exaflop per second by the end of the decade
- 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

So, how do we *study* these beasts?

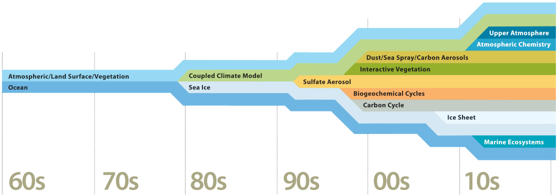
Computational Science of Computer Systems

My Research Field: Methodologies of Experimentation

- Goal: assess the performance and correctness of large-scale computer systems
- Question: Are we really producing scientifically sound results?
- Main contribution: SimGrid, a simulator of large-scale computer system

My approach: I am a physicist

- 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

Performance Study $\rightsquigarrow$ Experimentation

- Maths: Often not sufficient to fully understand these systems

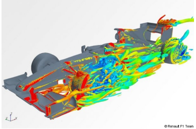
Correctness Study $\rightsquigarrow$ Formal Methods

- Tests: Unable to provide definitive answers

Assessing Distributed Applications

Performance Study ⇝ Experimentation

- Maths: Often not sufficient to fully understand these systems



By L. 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)*

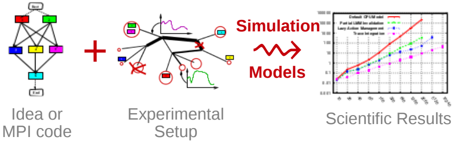
Correctness Study ⇝ Formal Methods

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

Simulating Distributed Systems

Simulation: Fastest Path from Idea to Data

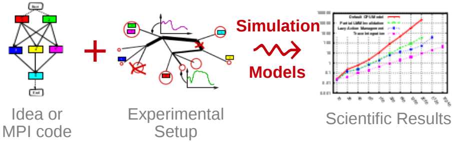
- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, don't fiddle with technical subtleties (yet)



Simulating Distributed Systems

Simulation: Fastest Path from Idea to Data

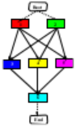
- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, don't fiddle with technical subtleties (yet)



Simulation: Easiest Way to Study Distributed Applications

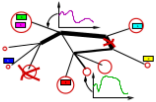
- Everything is actually centralized: Partially mock parts of your protocol
- No heisenbug: (Simulated) time does not change when you capture more data
- Clairevoyance: Observe every bits of your application and platform
- High Reproducibility: No or very few variability
- Capacity planning: Can we save on component? *What if* network were faster
- Don't waste resources to debug and test (up to 50% on some production infra)

Simulation Challenges



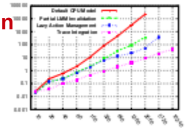
Idea or
MPI code

+

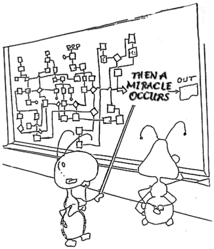


Experimental
Setup

Simulation
⇝
Models



Scientific Results



Challenges for the Tool Makers

- Validity: Get realistic results (controlled experimental bias). That's hard.
- Scalability: *Fast enough* and *Big enough*
- Tooling: runner, post-processing, integrated lab notes

Major Components of any Simulation-based Experiment

- An observation of your application: either a trace, prototype or live application
- A configuration describing the experimental settings
- Models of your platform: CPU, Network, Disk, any other relevant resource

SimGrid: Versatile Simulator of Distributed Apps

Scientific Instrument

- Versatile: Grid, P2P, IaaS Clouds, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Ready to use: Integrated to Debian/Ubuntu, self-contained Jar, win installer

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

Open Project with a Large Community

- Community-driven: 30 contributors (5 not affiliated), 5 contributed tools, GPL
- Impact: 120 publications (110 distinct authors, 5 continents), 4 PhD
- Started in 1998 at UCSD; Now collab accross many individuals and institutions
- 7 partners, 20+ researchers (CNRS, Universities, Inria)
- Public funding (≈ 3M€ ANR/Inria); Community based (User days, hackfests)

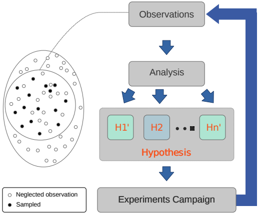
Simulation Validity

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
- Dimemas: aim at performance trends and bottleneck identification

SimGrid: 10-years effort on validity

- Same methodology than physicist: try to (in)validat our models
- Observe, analyze, hypothesis, test

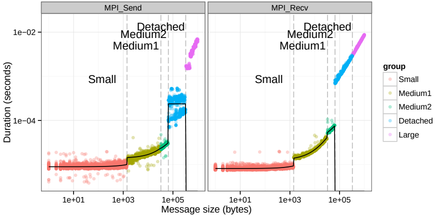


SimGrid provides several Network Models

- 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

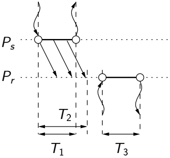
SimGrid Network Model

Measurements

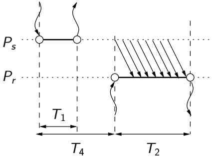


Model hybridizing LogP...

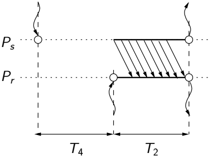
Asynchronous ($k \leqslant S_a$)



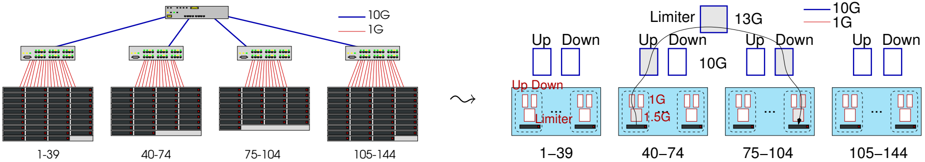
Detached ($S_a < k \leqslant S_d$)



Synchronous ($k > S_d$)



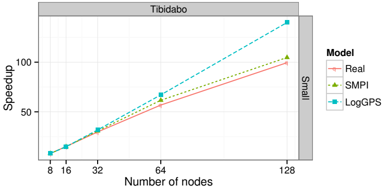
...and Fluid model: account for contention and network topology



SimGrid Validity Limits

Sometimes, it work rather well

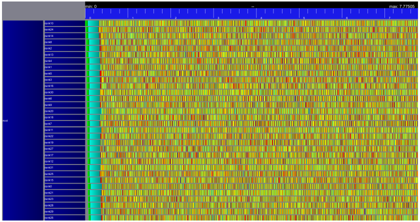
App: BigDFT (physics)
Host: Tibidabo (ARM + Ethernet 10G)

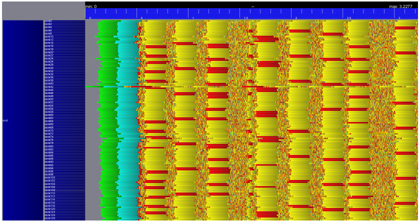


Sometimes, Simulation sucks

- Model limits, Bad instanciation, Applicative model faulty

Sometimes, Reality sucks





- NAS PB benchmark. Left: simulation; Right real execution
- Discrepancy: Reality experiences timeouts that are probably due to TCP RTO

Agenda

- Introduction
- Modern Large Computing Facilities
- Computational Science of Computer Systems (CS^2)
- Simulation Models
- Dynamic Verification of Distributed Applications
- Conclusion

Assessing the Correctness of HPC codes?

Writing Distributed Apps is notoriously difficult, but:

The Good Old Days

- MPI codes circumvented the difficulty with rigid communication patterns

- Correctness established through testing
- Only performance matters anyway:
  - Most prefer a fast code that rarely fail-stop to a slow code that always work
  - (at least, that's my feeling for most of the numerical applications)

These Days are Now Over

- But rigid patterns do not scale! We now have to release the grip
- But this is dangerous! We now have to explicitly seek for correctness

Slowly, old ignored problems resurface. . .

Model Checking and Dynamic Verification

These are Automated Formal Methods

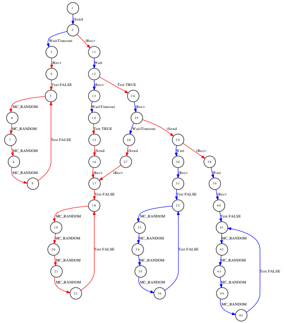
- Try to assess the correctness of a system by actively searching for faults
- If you find a fault, then you have something to work on
- If don't find any after an exhaustive search, correctness experimentally proved
- Dynamic Verification: Model Checking applied to real applications

Model Checking and Dynamic Verification

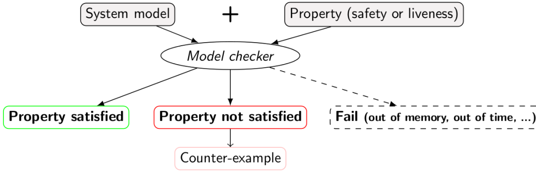
These are Automated Formal Methods

- Try to assess the correctness of a system by actively searching for faults
- If you find a fault, then you have something to work on
- If don't find any after an exhaustive search, correctness experimentally proved
- Dynamic Verification: Model Checking applied to real applications

Exhaustive Exploration



Model Checking: the Big Idea



- My preferred outcome: a counter-example
  If not, I fear my property to be wrongly expressed
- We tend to bug finding, not certification

Formal Properties

Safety Properties

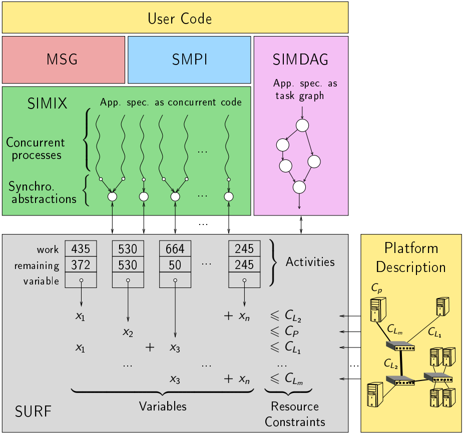
- "A given bad behavior never occurs"
- Can be expressed as boolean (assertion): no deadlock, $x \neq 0$, ...
- Work on all states separately
- Counter example: a faulty state

Liveness Properties

- "An expected behavior will happen in all cases"
- Example: Any process that asks a resource will obtain it eventually
- Must be expressed in a temporal logic such as CTL (safety ones *could* too)
- Work on execution path
- Counter example: an infinite path (ie, a cycle) that violates the property

Liveness properties are much more challenging to verify in practice

SimGrid and SMPI

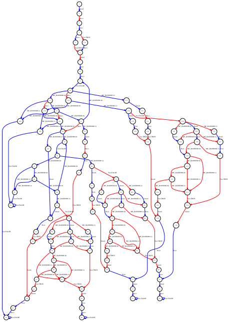


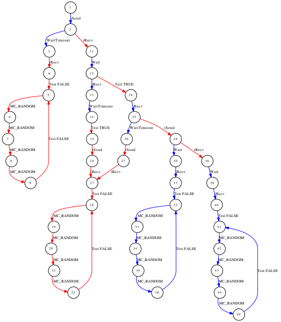
- SMPI can run complex C/C++/Fortran applications on top of SimGrid
- Let's leverage this unconventional virtualization layer for verification!
- + collective code scavenging $\rightsquigarrow$ verify even runtime's collectives

SimGridMC: Formal Methods in SimGrid

Verify any application that would run in SimGrid

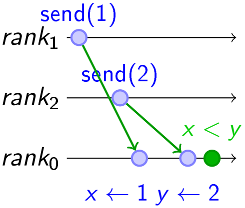
- Replace the simulation kernel underneath with a model checker
- Tests all causally possible orders of events to dynamically verify the app
- Reuse the mediation mechanism that base the simulator
- System-level checkpoints the app to then rewind and explore another path
- Works with SMPI, and MSG (our simple API for the study of CSP algorithms)



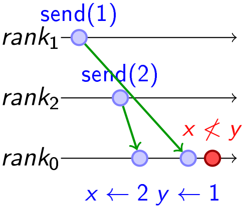


Example: Out of order receive

- Two processes send a message to a third one
- The receiver expects the message to be in order
- This may happen. . . or not




```
if (MPI_rank() == 0) {
  MPI_Recv(&x , MPI_ANY_SOURCE);
  MPI_Recv(&y , MPI_ANY_SOURCE);
  MC_assert(x < y);
} else {
  MPI_Send (&rank , 0);
}
```




```
*************************
*** PROPERTY NOT VALID ***
*************************
Counter-example execution trace:
[(1)recver] iRecv (dst=recver, buff=(verbose only), size=(verbose only))
[(3)sender] iSend (src=sender, buff=(verbose only), size=(verbose only))
[(1)recver] Wait (comm=(verbose only) [(3)sender -> (1)recver])
[(1)recver] iRecv (dst=recver, buff=(verbose only), size=(verbose only))
[(2)sender] iSend (src=sender, buff=(verbose only), size=(verbose only))
[(1)recver] Wait (comm=(verbose only) [(2)sender -> (1)recver])
```

Mitigating the State Space Explosion

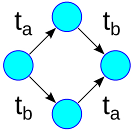
Many execution paths are redundant $\rightsquigarrow$ cut exploration when possible

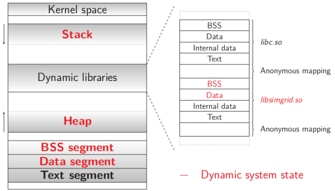
Dynamic Partial Ordering Reduction (DPOR)

- Works on histories: test only one transitions' interleaving if independent
- *Independence theorems:* Local events are independent; iSend+iRecv also; ...
- Must be conservative (exploration soundness at risk!)
- It works well (for safety properties)

System-Level State Equality

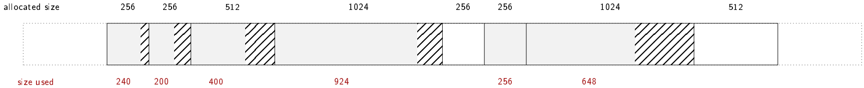
- Works on states: detect when a given space was previously explored
- Complementary to DPOR (but not compatible yet)
- Introspect the C/C++/Fortran app just like `gdb` (+some black magic)





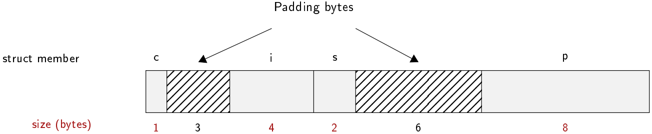
OS-level Challenges of State Equality Detection

- Memory over-provisioning

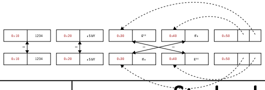


- Padding bytes: Data structure alignment

```
struct foo {
char c;
int i;
short s;
void *p;
}
```



- Irrelevant differences: system-level PID, fd, . . .
- Syntactic differences / semantic equalities:



Solutions

Issue	Heap solution	Stack solution
Overprovisioning	memset 0 (customized `mmalloc`)	Stack pointer detection
Padding bytes	memset 0 (customized `mmalloc`)	DWARF + libunwind
Irrelevant differences	Ignore explicit areas	DWARF + libunwind + ignore
Syntactic differences	Heuristic for semantic comparison	N/A (sequential access)

Some Results

Wild safety bug in our Chord implementation (≈ 500 lines of C)

- Simulation: bug on large instances only; MC finds small trace (1s with DPOR)

Mocked liveness bug

- Buggy centralized mutual exclusion: last client never obtains the CS
- About 100 lines – state snapshot size: 5Mib
- Verified with up to 7 processes (12,000 states, 9 minutes, 45Gb).

Verifying MPICH3 complience tests

- Looking for assertion failures, deadlocks and non-progressive cycles
- 6 tests; ≈ 1300 LOCs (per test) – State snapshot size: ≈ 4MB
- With no reduction: no test concluded in a few hours
- With state equality: Exhaustive exploration up to 10 procs, but no error found
- With memory compaction: use only dozen of Gb in RAM, not hundreds
- We verified several MPI2 collectives too ☺ (but all good so far ☹)

Verification of Protocol-wide Properties

Motivation

- Clever checkpoint algorithms exist, provided that the application is nice enough
- *On communication determinism in parallel HPC applications*, F. Cappello, A. Guermouche and M. Snir (2010)
  - Manual inspection of 27 HPC applications, seeking for such properties

Protocol-wide properties

- deterministic: On each node, send and receive events are always in same order
- send deterministic: $\forall$ node, send are always the same, no matter the recv order
- Not liveness, not even LTL: quantifies for all execution paths within property

Status report: we can verify such properties in SimGrid

- Explore one path to learn the communication order, deduce the property
- Enforce that this order holds on all other execution path
- We reproduced the conclusions of previous paper on several benchmarks
  - All good ☹

More on Formal Verification

We've built a really cool tool

- We can verify many unmodified MPI applications (C/C++/Fortran)
- State space reduction: DPOR or State equality (not together yet)
- Properties: safety, liveness or protocol-wide

Many remaining Research Leads

- Other reductions, HPC-specific properties, statistical model-checking, . . .
- Interactive tool to get gdb-like info on each state in the execution graph

We need more use cases

- We are done with all the ones provided by the practitioners we know
- We could make it even better with really relevant use cases
- We don't know what properties are relevant

Agenda

- Introduction
- Modern Large Computing Facilities
- Computational Science of Computer Systems (CS^2)
- Simulation Models
- Dynamic Verification of Distributed Applications
- Conclusion

Much more to say about SimGrid (too little time)

Hybrid Network Models

- Fluid model: model contention in steady state for large messages
- LogOP model: model intra-node delays and synchronization
- Also: MPI collectives, TCP (slow-start, cross-traffic), soon IB

Realistic Emulation

- SMPI: Study real MPI applications within SimGrid
- Simterpose: Study real arbitrary applications (ongoing)

High Performance Simulation

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

Formal Verification of Distributed Apps

- Safety, Liveness or CTL properties, with DPOR or state equality

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; AGPL
- 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 ;)

`apt-get install simgrid` now! (or get the jarfile)