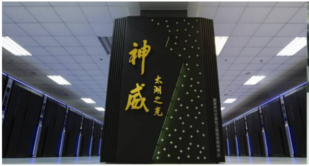
Establishing a Mature Simulator of IT Systems
Scientific, Technical and Community considerations

Martin Quinson (ENS Rennes / Inria, France)
joint work with **many** colleagues over 15 years.

Séminaire DGD-T Inria
May 23., 2018

Modern Large Scale Distributed Systems

Huge Systems

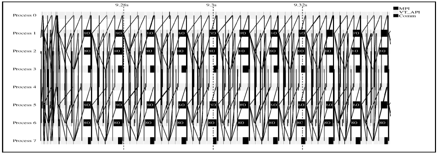


#1 Taihu Light
10,649,600 cores
125 Tflops, 15MW

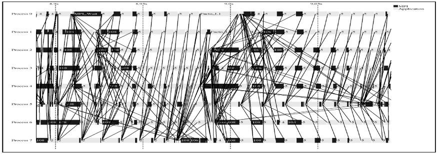
#2 Tianhe 2
3,120,000 cores
56 Tflops, 18MW

#3 Piz Daint
361,760 cores
25 Tflops, 2MW

Complex Applications



Rigid, Regular, Hand-tuned Comm Patterns



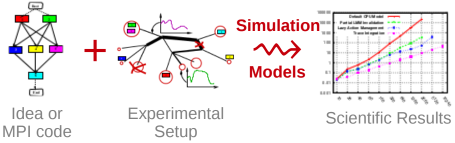
Dynamic, Irregular (task-based?)

How do we study these beasts?

Simulating Distributed Systems

Simulation: Fastest Path from Idea to Data

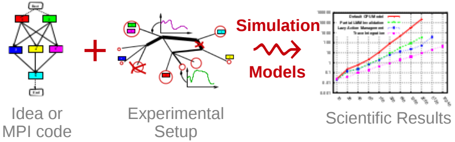
- Test your scientific idea with a fast and confortable scientific instrument



Simulating Distributed Systems

Simulation: Fastest Path from Idea to Data

- Test your scientific idea with a fast and confortable scientific instrument



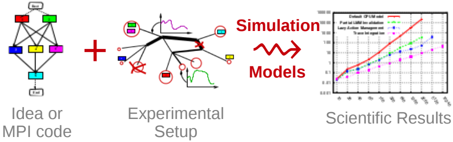
Simulation: Easiest Way to Study Real Distributed Systems

- Centralized and reproducible setup. Don't waste resources to debug and test
- No Heisenbug, full Clairevoyance, High Reproducibility, *What if* studies

Simulating Distributed Systems

Simulation: Fastest Path from Idea to Data

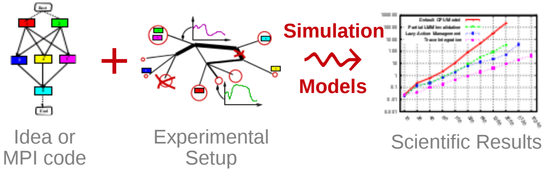
- Test your scientific idea with a fast and confortable scientific instrument

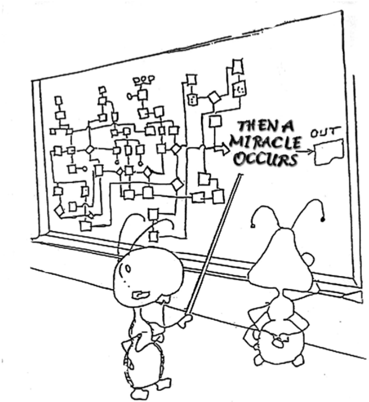


Simulation: Easiest Way to Study Real Distributed Systems

- Centralized and reproducible setup. Don't waste resources to debug and test
- No Heisenbug, full Clairevoyance, High Reproducibility, *What if* studies
- Also software/hardware co-design, capacity planning or hardware qualification

Methodological Challenges raised





Challenges

- Validity: Realistic results
- Scalability: Fast enough; Big enough
- Right Focus: Aligned with users concerns

Flourishing State of the Art

- Each group / student build its own tool
  - Short lived, Narrow focus, Improvable
- Some very good domain-specific tools (HPC)

SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

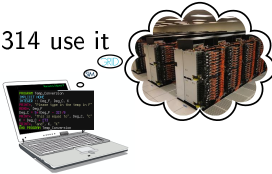
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science $\leadsto$ Predictive Power
- Architectured as an OS $\leadsto$ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

Community

- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)



SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

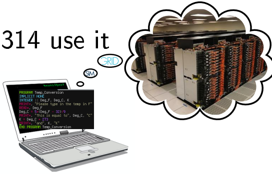
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science ⇝ Predictive Power
- Architectured as an OS ⇝ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

Community

- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)



The Many Interfaces of SimGrid

SMPI: Reimplementation of MPI on top of SimGrid

- Complex in C/C++/Fortran applications *virtualized* out of the box

MSG: legacy interface for Concurent Sequential Processes

- Goal: ease the study of distributed algorithms (C or Java)
- Initially for distributed scheduling, used in many other contexts since 2005
- **Our main interface** is slowly getting crippled (backward compat when possible)

SimDag: legacy interface for DAG scheduling

- Goal: ease the study of centralized algorithms, since 1998

S4U: Future interface for algorithms

- Currently under development toward SimGrid 4, already usable (C++ or C)

BYOS: Build Your Own Simulator

- PSG Project: PeerSim interface implemented on top of SimGrid
- Wrench Project: Workflow Management System Simulation Workbench

SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

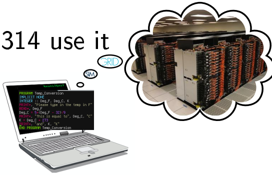
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science ⇝ Predictive Power
- Architectured as an OS ⇝ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

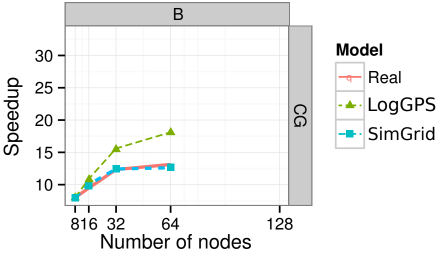
Community

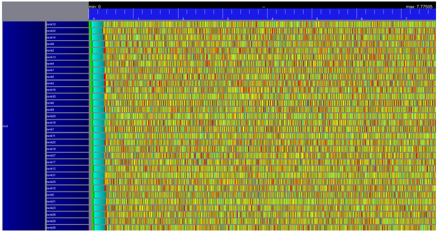
- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)



Validity Success Stories

unmodified NAS CG on a TCP/Ethernet cluster (Grid'5000)



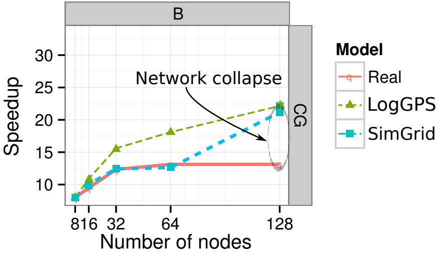


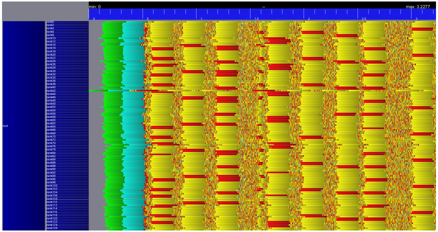
Key aspects to obtain this result

- Network Topology: Contention (large msg) and Synchronization (small msg)
- Applicative (collective) operations (stolen from real implementations)
- Instantiate Platform models (matching effects, not docs)
- All included in SimGrid but the instantiation, remains manual (for now)

Validity Success Stories

unmodified NAS CG on a TCP/Ethernet cluster (Grid'5000)





Discrepency between Simulation and Real Experiment. Why?

- Massive switch packet drops lead to 200ms timeouts in TCP!
- Tightly coupled: the whole application hangs until timeout
- Noise easy to model in the simulator, but useless for that very study
- Discrepancy between simulated and real-world is actually a real-world problem

Have we reached the Perfect Model yet?

What is the Perfect Model anyway?

- Detailed enough to be realistic
- Efficient enough for ultra fast simulations
- Abstracted enough so that I can reason about
- In short, that's the one I could give to my students and forget about

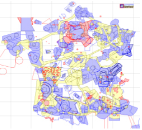
Perfect Model of France would be Perfect Map

Maps (and models) are abstractions

- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

Perfect Model of France would be Perfect Map



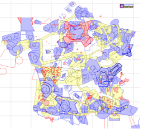



Maps (and models) are abstractions

- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

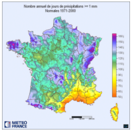
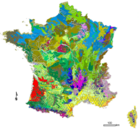
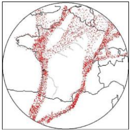
Perfect Model of France would be Perfect Map



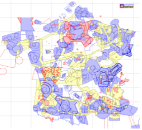
Maps (and models) are abstractions

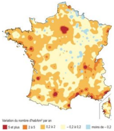
- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

Perfect Model of France would be Perfect Map

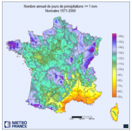
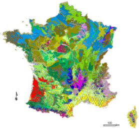
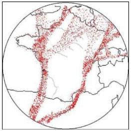


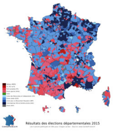



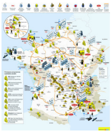
Maps (and models) are abstractions

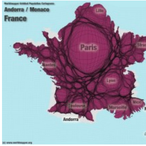
- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps





Perfect Model of Distributed Systems?

the one making your Study sound

If you study a theoretical P2P algorithm

- You could probably go for a super-fast constant-time model

If your study is a MPI application

- with TCP LAN, SMPI should do the trick (with correct instantiation)
- with InfiniBand and/or GPUs, you need our still ongoing models

If you work on a TCP variant

- then you need a packet-level simulator such as NS3

If your study WAN-interconnected Set Top Boxes

- SMPI model not suited! Impossible to instanciate, validated only for MPI
- Vivaldi model intended for that kind of studies

In any case, assess the validity & soundness

SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

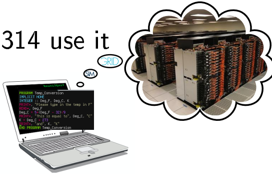
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

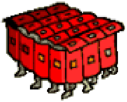
- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science ⇝ Predictive Power
- Architectured as an OS ⇝ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

Community

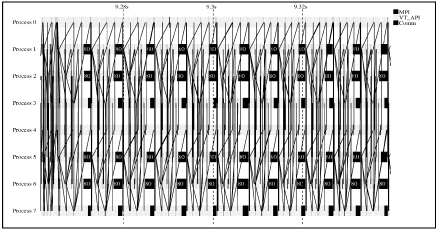
- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)



Writting Correct Distributed Applications

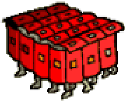


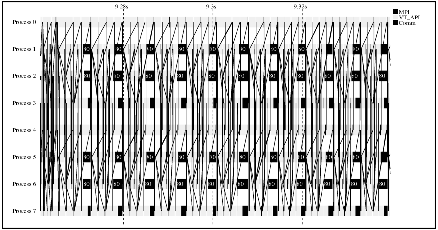
- Classical Solution: Proof of algorithms
- Pessimistic Solution: Lower performance expectations
- Optimistic Solution: Eventually Consistent
- HPC Solution: Rigid, Regular, Hand-tuned Communication Patterns

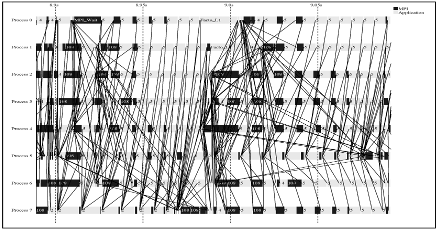


Writting Correct Distributed Applications

- Classical Solution: Proof of algorithms
- Pessimistic Solution: Lower performance expectations
- Optimistic Solution: Eventually Consistent
- HPC Solution: Rigid, Regular, Hand-tuned Communication Patterns
- Large-Scale Hybrid Machines: Dynamic, Irregular (task-based?)

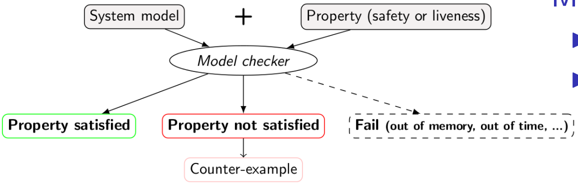






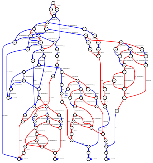
Verification: must explore **all possible execution paths**

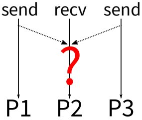
Formal Methods in Mc SimGrid



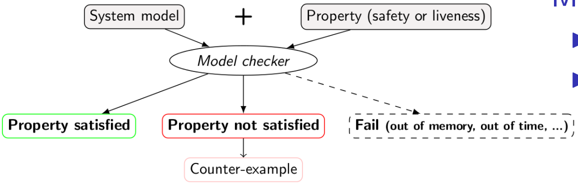
Model Checking

- Exhaustively search for faults
- Requires an accurate model





Formal Methods in Mc SimGrid

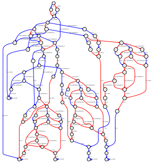


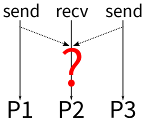
Model Checking

- Exhaustively search for faults
- Requires an accurate model

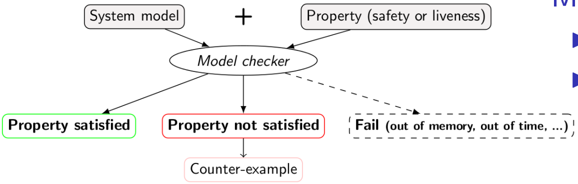
Dynamic Verification: similar idea, applied to source code

- Mc SimGrid: Live, virtualized execution
  No static analysis (yet), no symbolic execution
- On Indecision Points: checkpoint, explore, rollback





Formal Methods in Mc SimGrid

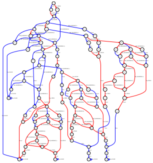


Model Checking

- Exhaustively search for faults
- Requires an accurate model

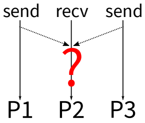
Dynamic Verification: similar idea, applied to source code

- Mc SimGrid: Live, virtualized execution
  No static analysis (yet), no symbolic execution
- On Indecision Points: checkpoint, explore, rollback



Execution Model in Mc SimGrid

- Mono-threaded MPI applications (CSP)
- Point-to-Point semantic: Configurable (paranoid / permissive)
- Collective semantic: Implementations of MPICH3, OpenMPI



Mc SimGrid Overview

Mc SimGrid: Dynamic Verification of MPI applications

- Unmodified C/C++/Fortran MPI applications
- Early stage, but already functional: Safety, Liveness, Send-determinism
- Reductions: DPOR and State Equality
- Scale to a few processes only, but exhaustive testing

State of the Art

- Many testing tools (MUST): not exhaustive nor sound
- Symbolic execution (TASS, CIVL): complementary to our work
- Dynamic verification (ISP, DAMPI at U. Utah)
  - PMPI proxy at runtime to delay communications to guide execution
  - Works for safety, but not applicable to liveness (state equality)

Ongoing Works

- Improve DPOR by using Event Unfolding structures (IPL PhD)
- Convert checkpoints taken on OpenMPI into SimGrid runs (IPL Post-doc)
- Static Analysis to improve Dynamic State Equality Detection (IPL collab)

SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

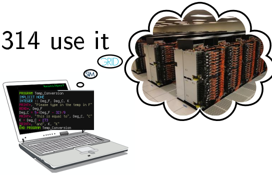
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science ⇝ Predictive Power
- Architectured as an OS ⇝ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

Community

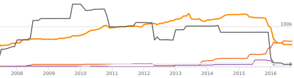
- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)

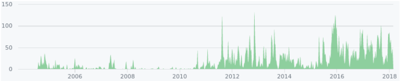


Technical Considerations

Complex and Dynamic Code Base

- Only 100k sloc, but complex due to versatile efficiency + formal verification
- Implemented in C++/C (+ assembly); Bindings: Java, Lua and Fortran
- Active project: commits every day by ≈ 6 commiters, 4 releases a year
- Ongoing full rewrite in C++ along with *Release soon, Release often*

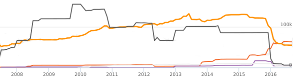


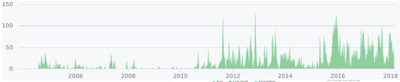


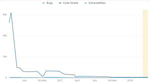
Technical Considerations

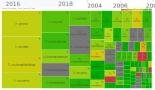
Complex and Dynamic Code Base

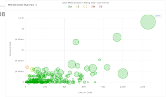
- Only 100k sloc, but complex due to versatile efficiency + formal verification
- Implemented in C++/C (+ assembly); Bindings: Java, Lua and Fortran
- Active project: commits every day by ≈ 6 commiters, 4 releases a year
- Ongoing full rewrite in C++ along with *Release soon, Release often*











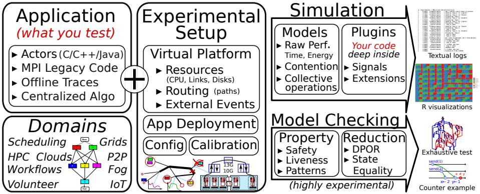
Well Tested

- 740 integration tests (full simulations w/ timings), 10k units. Coverage: 80%
- Inria CI (Jenkins), Travis, Appveyor: we use the clouds!
- Each commit: Linux (9 versions), FreeBSD, Windows (native/ubuntu), Mac
  Compiler: gcc/clang/ICC; Arch: 64bits/32bits (+ 12 Debian Builders)
- Nightly: Valgrind, clang sanitizers, clang static analyzer, codacy, SonarQube
  Tested External Projects: StarPU, BigDFT and ExaScale Project's Proxy Apps
- Ongoing/Planned: (better) Performance Testing; Toward a unique config

Toward the Users

Documentation

- Lot of slides: 101 presentations and tutorials
- Quick start tutorials for MSG, SimDag and MPI (planned: s4u)
- Many examples, on most features (used as integration tests)
- User documentation still incomplete; Reference guide re-org ongoing



- SMPI CourseWare: programming assignments (planned: MOOC)

Large Diffusion

- Open Source: LGPL (non viral)
- Source archive: Inria gforge ⇝ GitHub (Inria's GitLab: too closed)
- Binaries: Integrated in Debian and Ubuntu (Java binary jar on webpage)

Development and Valorization Strategy

Goal: Ensure the Software Sustainability

- Beyond scientific projects (ANR, IPL): best transfer strategy = open access
- Engineering tasks currently handled by scientists (IJD not enough)

Cultivate our garden: simplify everything to grow further

- Time consuming but rewarding: huge competitive advantage in science

Inria ADT engineer 2017-2019: Simgrid As A Platform (SaaP)

- Next Generation API (SimGrid 4) $\rightsquigarrow$ Build Your Own Simulator
- Add callbacks for plugins, rework modularity for power users
- Improve examples and documentation for newcomers
- Provide compatibility layers to other simulators (PeerSim, DCSim)

Building a Community

Communication and Animation

- SimGrid User Days: Welcome newcomers & Take feedback since 2010
- Scientific tutorials, Booth at SuperComputing, Presence at other confs
- Missing instrument: Companies Courses

Preliminary Industrial Contacts

- CERN: currently testing the LHC DataGrid before production
- Intel/KAUST: internal project (est. at SC'17)
- Octo: dimensionning Ceph infrastructures for their clients (attempt)
- Bull: used internally, but not officially yet :)
- Amazon/Nice: very preliminary contacts for dimensionning, service to clients
- My dream: make open-source IT infra (Samba, Ceph) testable with SimGrid
- Possible Income: subscription of 6-8 supporting institutions/companies

Toward Education

- Teach now the researchers and engineers of tomorrow to SimGrid
- Done: SMPI CourseWare, PeerSimGrid; Ongoing: Cloud, Wrench and more?

SimGrid Governance

Core Team

- A. Legrand (Polaris): Kernel + Models + Reproducibility
- M. Quinson (Myriads): Model Checking + Software Architect
- F. Suter (Avalon): DAGs + Storage + Evangelism
- H. Casanova (UH Manoa): original project developper; now Happy User

Strengths

- Very complementary skills, well identified non competing interests
- Long collaboration history (15 years)
- Daily interactions (IRC, mail, phone, visits, etc)
- Fair distribution of funded project management
- Easy agreement on important decisions
- Variable respective contribution to the project over time

Weakness

- Very informal, with no issue handling procedures

Community Building

Vision: Doing Good Science about Distributed IT Systems

- Become the Valgrind of Distributed IT Systems, and push Open Science

GitHub Criteria

- Have Basics, Readme, Contributing
- Missing Code of Conduct

Core Infrastructure Initiative Criteria

- 97% of basis: Missing https website and md5sum of package
- 65% of Silver, only. Missing:
  - Project: Governance Model, Code of Conduct.
  - Doc: Roadmap, Technical Overview, Upgrade Path, Accessibility.
  - Code: Regression tests for the bugs
  - Security: Vulnerability Response, Signed Releases+VCS, etc.
- 39% of Gold, only. Missing:
  - 2 unassociated significant contributors, List of small tasks, 2FA
  - Systematic Code Review, 90% coverage, Security Audit

SimGrid: Versatile Simulator of Distributed Apps

Put a Scientific Instrument on your Laptop and Do Good Science

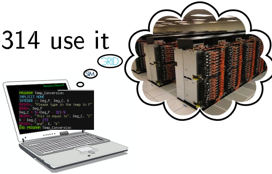
- Joint Project since 1998, mostly from French institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

Key Strengths

- Usability: Fast, Reliable, User-oriented APIs, Visualization
- Performance Models validated with Open Science ⇝ Predictive Power
- Architectured as an OS ⇝ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users

Community

- Scientists: 500+ publications only cite it, 58 extend it, 314 use it
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, Bull)
- Open Source: external Power Users (fixes & models)



Question slides

IPL HAC-SPECIS (2016-2020)

Inria Project Lab $\approx$ 1 postdoc and 1 PhD student per year for 3-4 years

Project Partners

8 Inria Teams (verification^{+}, performance evaluation$^{\triangle}$, HPC*) + CEA*

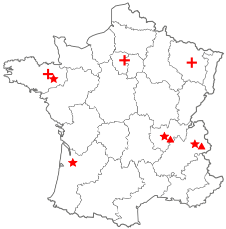
Rhône Alpes: AVALON* $^{\triangle}$, POLARIS* $^{\triangle}$ + CEA*

Rennes: MYRIADS* $^{+}$, SUMO^{+}

Bordeaux: HIEPACS*, STORM*

Paris: MEXICO^{+}

Nancy VERIDIS^{+}



Context and Objectives

- Rigid communication patterns are not scalable enough:
- HPC apps become adaptive, lock-free, with complex optimizations/scheduling

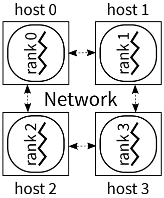
- **Research Question:** Joint Study of Performance **AND** Correctness
- Goal: *bridge the gap between communities*

Virtualizing MPI Applications with SimGrid

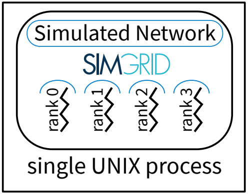
SMPI: Reimplementation of MPI on top of SimGrid

- Computations emulated; Communications simulated
- Complex C/C++/F77/F90 apps run out of the box
  - 23 out of 30 Exascale Project's proxy apps supported (others: 5 extra deps, 2 unsupported MPI calls)
- MPI 2.2 partially covered ($\approx$ 160 primitives supported)
  - No MPI-IO, MPI3 collectives, spawning ranks, . . .
  - Monothreaded applications, no pthread nor OpenMP

MPI Applications are *folded into* a single process



Real Settings

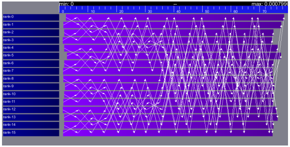
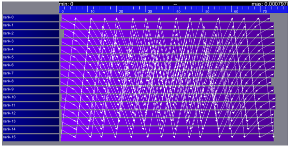
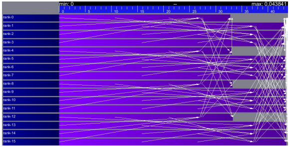


SimGrid Simulation

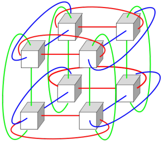
SimGrid Modeling of MPI

MPI Collectives

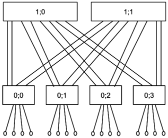
- SimGrid implements more than 120 algorithms for the 10 main MPI collectives
- Selection logic from OpenMPI, MPICH can be reproduced

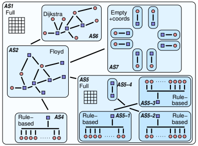
HPC Topologies



Torus



Fat-trees



Hierarchies of ASes

But also

- External load (availability changes), Host and link failures, Energy (DVFS)
- Virtual Machines, that can be migrated; Random platform generators

What Kind of Properties can be Verified?

Safety Properties: "A given bad behavior never occurs"

- e.g.: any assertion (x != 0, no deadlock)
- Verified on each state separately
- Counter example: a faulty state

Liveness Properties: "An expected behavior will happen in all cases"

- e.g.: Any request will eventually be fulfilled; No non-progression cycle
- Verified on a full execution path
- Counter example: a cycling execution path that violates the property

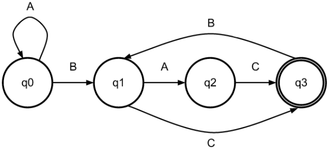
Comm Patterns: "It exists a pattern that is the same for all exec paths"

- e.g.: send-deterministic (local sending order is always the same)
- Work on all execution paths
- Counter examples: two paths exhibiting differing communication patterns

Checking Liveness Properties

Enforce property ϕ

- Search for a counter-example, ie a run of the system satisfying $\neg\phi$
- Counter examples are infinite $\rightsquigarrow$ Build the Büchi Automaton of $\neg\phi$



- Ensure that Application $\times$ Bucchi($\neg\phi$) is empty (no accepted run)
- State Equality is crucial to detect cycles

Current state in Mc SimGrid

- Working in our tests (although fragile: equality is based on heuristics)
- We are looking for more domain-specific interesting properties

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 Mc 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
  - NAS Parallel Benchmarks NPB 3.3 (5 kernels)
  - CORAL Benchmark codes
  - NERSC-8/Trinity Benchmarks

Mitigating the State Space Explosion

The exploration process often fails to complete

- Too many states to explore, not enough time and/or memory
- Mc SimGrid provides two reductions techniques

Dynamic Partial Ordering Reduction (DPOR)

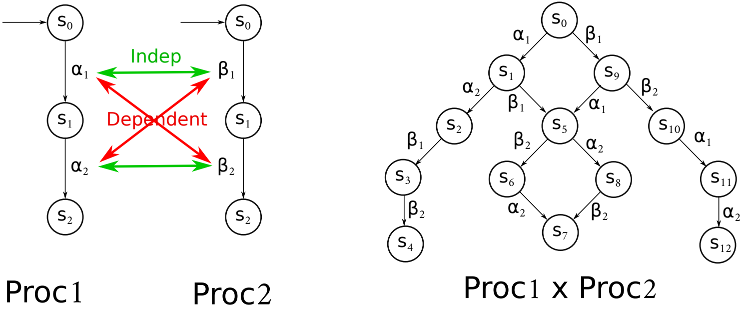
- Avoid re-exploring equivalent interleavings
- Don't explore all interleavings of local executions: they are equivalent

System-Level State Equality

- Detect when a given state was previously explored

Partial Ordering Reduction (DPOR)

- Avoid re-exploring Mazurkiewicz traces (don't permute independent events)



- McSimGrid: iSend and iRecv are independent, etc.
- Dynamic Partial Ordering Reductions take advantage of runtime knowledge
- Many techniques (sleep sets, ample sets) are hard to understand & get right
- Ongoing work: reimplement our DPOR using Event Unfolding Structures

But what are the transitions in Mc SimGrid?

Transition = atomic block of code between Indecision Points

- Test all interleavings of the shared state (mem+network) modifications
- Transition = (some local code +) **one** shared state's change

But what are the transitions in Mc SimGrid?

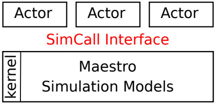
Transition = atomic block of code between Indecision Points

- Test all interleavings of the shared state (mem+network) modifications
- Transition = (some local code +) **one** shared state's change

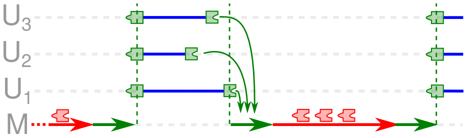
Implementation: SimGrid is an Operating System

- Actors must use simcalls to modify the shared state
- First introduced for parallel simulation, but crucial to dynamic verification

Functional View

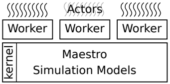


Temporal View

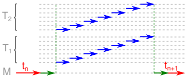


Going parallel

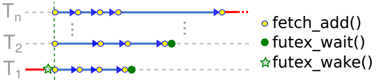
- More actors than cores ⇝ Worker Threads that execute co-routines



Functional View



Temporal View



Ideal Algorithm

Mitigating the State Space Explosion

The exploration process often fails to complete

- Too many states to explore, not enough time and/or memory
- Mc SimGrid provides two reductions techniques

Dynamic Partial Ordering Reduction (DPOR)

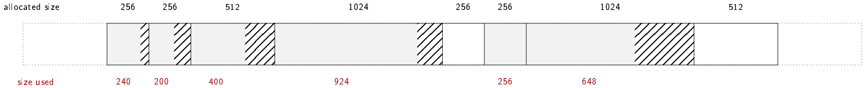
- Avoid re-exploring equivalent interleavings
- Don't explore all interleavings of local executions: they are equivalent

System-Level State Equality

- Detect when a given state was previously explored
- Introspect the application state similarly to gdb
- Also with Memory Compaction

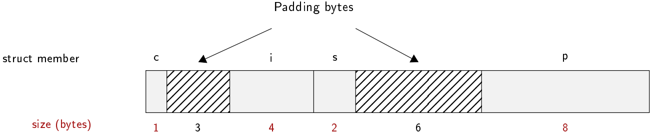
OS-level State Equality Detection

- Memory over-provisioning

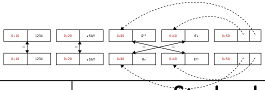


- Padding bytes: Data structure alignment

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



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



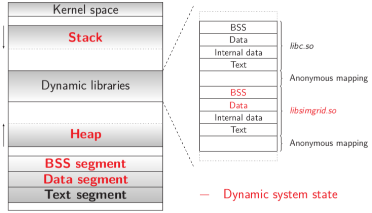
Solutions

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)

Applicative State in Mc SimGrid

We work at system level

- Target = legacy MPI apps
- Stack: where maestro lives
- Heap: shared between actors + actors stacks
- BSS+Data: private copy for each actor
- Network state is within libsimgrid data

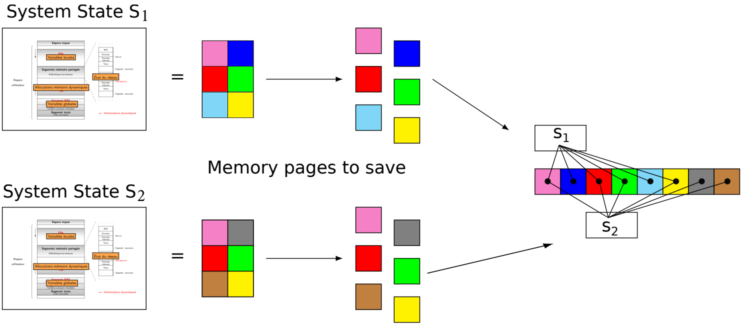


How to privatize the BSS+data

- (this is required to fold MPI processes anyway)
- Source-to-Source: turn globals into arrays of locals
- Compiler's pass: move globals into TLS area
  changes toolchain (no `icc`) $\rightsquigarrow$ alters SEBs (as any previous solution)
- GOT injection: rewrite the ELF symbol table when switching contextes
  static variables are not part of the GOT unfortunately
- mmap of bss+data segments: preserves SEBs but forces sequential exec
- dlopen tricks: compile app with -fPIE, dlopen() it many times

Memory Compactions

We save literally thousands of states



- Very few modification between states in practice
- First fast hash function to distinguish new pages, then byte-wise equality
- Combines nicely with State Equality Detection (but complex implementation)

Evaluation

Verified small applications

- MPI2 collectives, MPICH3 test suite, Benchmarks (NAS, CORAL, NERSC)
- Safety, Liveness (no non-progressive cycle), Send-determinism

Results

- Without reduction, only scales up to 2 to 6 processes in 24h
- Reductions (when usable) and Memory Compaction goes a bit further
- Not exactly ExaScale, but exhaustively at small size already useful

Found bugs

- The one we intentionally added to the code
- Our own implementation of the Chord protocol (not in MPI)
- But no wild bugs in MPI yet :(

Verification of some MPICH3 unit tests

- Looking for assertion failures, deadlocks and non-progressive cycles
- Exhaustive exploration, but no error found
- ≈ 1300 LOCs (per test) – State snapshot size: ≈ 4MB

Application	#P	Stateless exploration		Stateful exploration		
		# States	Time	# States	Time	Memory
sendrecv2	2	> 55 millions	> 6h	936	13s	2GB
	5	-	-	2 284	43s	5.4GB
	10	-	-	3 882	2m	11GB
bcastzerotype	5	> 12 millions	> 1h	2 474	41s	3.1GB
	6	-	-	17 525	5m	19GB
coll4	4	> 100 millions	> 24h	29 973	20m	38GB
	5	-	-	> 150 000	> 4h	> 200GB
groupcreate	5	> 10 millions	> 1h30	2 217	38s	2.8GB
	7	-	-	71 280	24m	62GB
dup	4	> 57 millions	> 5h	4 827	1m20	6.5GB
	5	-	-	75 570	49m	87GB

- We verified several MPI2 collectives too: all good so far ☹