Experimenting HPC Systems with Simulation[1]

Martin Quinson (Nancy University, France)
et Al.

Caen, June 28 2010
(HPCS/IWCMC 2010 Tutorial)

[1]Partially funded by ANR 08 SEGI 022 Project.

Scientific Computation Applications

Classical Approaches in science and engineering

1. Theoretical work: equations on a board
2. Experimental study on an scientific instrument

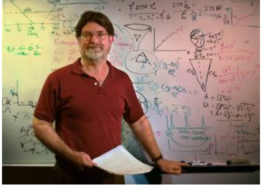
That's not always desirable (or even possible)

- Some phenomenons are intractable theoretically
- Experiments too expensive, difficult, slow, dangerous

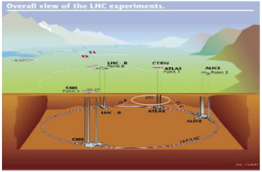
The third scientific way: *Computational Science*

3. Study in silico using computers
   Modeling / Simulation of the phenomenon or data-mining

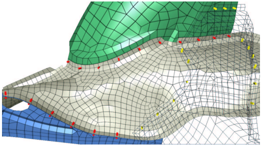
⇝ High Performance Computing Systems



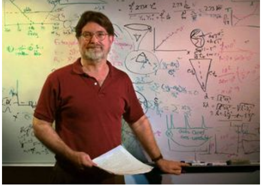
Georges Smoot
Physics Nobel Price 1996



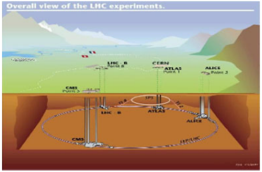
Large Hardron Collider



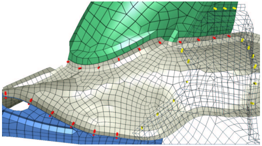
Scientific Computation Applications



Georges Smoot
Physics Nobel Price 1996



Large Hardron Collider



The third scientific way: *Computational Science*

3. Study in silico using computers
   Modeling / Simulation of the phenomenon or data-mining

$\rightsquigarrow$ High Performance Computing Systems

These systems deserve very advanced analysis

- Their debugging and tuning are technically difficult
- Their use induce high methodological challenges
- Science of the *in silico science*

Large-Scale Distributed Systems Science?

Requirement for a Scientific Approach

- Reproducible results
  - You can read a paper,
  - reproduce a subset of its results,
  - improve
- Standard methodologies and tools
  - Grad students can learn their use and become operational quickly
  - Experimental scenario can be compared accurately

Current practice in the field: quite different

- Very little common methodologies and tools
- Experimental settings rarely detailed enough in literature

Large-Scale Distributed Systems Science?

Requirement for a Scientific Approach

- Reproducible results
  - You can read a paper,
  - reproduce a subset of its results,
  - improve
- Standard methodologies and tools
  - Grad students can learn their use and become operational quickly
  - Experimental scenario can be compared accurately

Current practice in the field: quite different

- Very little common methodologies and tools
- Experimental settings rarely detailed enough in literature

Purpose of this tutorial

- Present "emerging" methodologies and tools
- Show how to use some of them in practice
- Discuss open questions and future directions

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
  - Associated Tools
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

Distributed Systems: Analytical or Experimental?

Analytical works?

- Some purely mathematical models exist
- ☺ Allow better understanding of principles in spite of dubious applicability
  impossibility theorems, parameter influence, ...
- ☹ Theoretical results are difficult to achieve
  - Everyday practical issues (routing, scheduling) become NP-hard problems
    Most of the time, only heuristics whose performance have to be assessed are proposed
  - Models too simplistic, rely on ultimately unrealistic assumptions.

$\Rightarrow$ One must run experiments

$\rightsquigarrow$ Most published research in the area is experimental

- In vivo: Direct experimentation
- In silico: Simulation
- In vitro: Emulation

Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
    - In vivo approach (direct experimentation)
    - In silico approach (simulation)
    - In vitro approach (emulation)
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
  - Associated Tools
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

In vivo approach to HPC experiments (direct experiment)

- ☺ Eminently *believable* to demonstrate the proposed approach applicability

In vivo approach to HPC experiments (direct experiment)

- ☺ Eminently *believable* to demonstrate the proposed approach applicability
- ☹ Very time and labor consuming
  - Entire application must be functional
  - Parameter-sweep; Design alternatives
- ☹ Choosing the right testbed is difficult
  - My own little testbed?
    - ☺ Well-behaved, controlled,stable
    - ☹ Rarely representative of production platforms
  - Real production platforms?
    - Not everyone has access to them; CS experiments are disruptive for users
    - Experimental settings may change drastically during experiment (components fail; other users load resources; administrators change config.)
- ☹ Results remain limited to the testbed
  - Impact of testbed specificities hard to quantify ⇒ collection of testbeds...
  - Extrapolations and explorations of “what if” scenarios difficult (what if the network were different? what if we had a different workload?)
- ☹ Experiments are uncontrolled and unrepeatable

  No way to test alternatives back-to-back (even if disruption is part of the experiment)

In vivo approach to HPC experiments (direct experiment)

- ☺ Eminently *believable* to demonstrate the proposed approach applicability
- ☹ Very time and labor consuming
  - Entire application must be functional
  - Parameter-sweep; Design alternatives
- ☹ Choosing the right testbed is difficult
  - My own little testbed?
    - ☺ Well-behaved, controlled,stable
    - ☹ Rarely representative of production platforms
  - Real production platforms?
    - Not everyone has access to them; CS experiments are disruptive for users
    - Experimental settings may change drastically during experiment (components fail; other users load resources; administrators change config.)
- ☹ Results remain limited to the testbed
  - Impact of testbed specificities hard to quantify ⇒ collection of testbeds...
  - Extrapolations and explorations of “what if” scenarios difficult (what if the network were different? what if we had a different workload?)
- ☹ Experiments are uncontrolled and unrepeatable

  No way to test alternatives back-to-back (even if disruption is part of the experiment)

Difficult for others to reproduce results

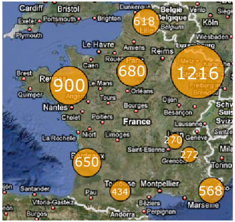
even if this is the basis for scientific advances!

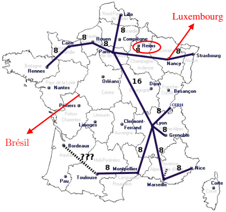
Example of Tools for Direct Experimentation

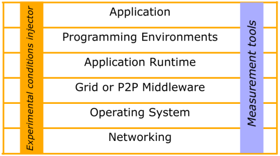
- Principle: Real applications, controlled environment
- Challenges: Hard and long. Experimental control? Reproducibility?

Grid'5000 project: a scientific instrument for the HPC

- Instrument for research in computer science (*deploy* your own OS)
- 9 sites, 1500 nodes (3000 cpus, 4000 cores); dedicated 10Gb links







Other existing platforms

- PlanetLab: No experimental control ⇒ no reproducibility
- Production Platforms (EGEE): must use provided middleware
- FutureGrid: future US experimental platform loosely inspired from Grid'5000

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world
  - Scalability: Actually usable by computers (fast enough)

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world
  - Scalability: Actually usable by computers (fast enough)
  - Tractability: Actually usable by human beings (simple enough to understand)

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state $\oplus$ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world
  - Scalability: Actually usable by computers (fast enough)
  - Tractability: Actually usable by human beings (simple enough to understand)
  - Instanciability: Can actually describe real settings (no magical parameter)

In silico approach to HPC experiments (simulation)

☺ Simulation solves some difficulties raised by *in vivo* experiments

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state ⊕ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world
  - Scalability: Actually usable by computers (fast enough)
  - Tractability: Actually usable by human beings (simple enough to understand)
  - Instanciability: Can actually describe real settings (no magical parameter)
  - Relevance: Captures object of interest

Simulation in Computer Science

Microprocessor Design

- A few standard "cycle-accurate" simulators are used extensively
  `http://www.cs.wisc.edu/~arch/www/tools.html`

⇒ Possible to reproduce simulation results

Networking

- A few established "packet-level" simulators: NS-2, DaSSF, OMNeT++, GTNetS
- Well-known datasets for network topologies
- Well-known generators of synthetic topologies
- SSF standard: `http://www.ssfnet.org/`

⇒ Possible to reproduce simulation results

Large-Scale Distributed Systems?

- No established simulator up until a few years ago
- Most people build their own "ad-hoc" solutions

  Naicken, Stephen *et Al.*, *Towards Yet Another Peer-to-Peer Simulator*, HET-NETs'06.

  From 141 P2P sim.papers, 30% use a custom tool, 50% don't report used tool

Simulation in Parallel and Distributed Computing

- Used for decades, but under drastic assumptions in most cases

Simplistic platform model

- Fixed computation and communication rates (Flops, Mb/s)
- Topology either fully connected or bus (no interference or simple ones)
- Communication and computation are perfectly overlappable

Simplistic application model

- All computations are CPU intensive (no disk, no memory, no user)
- Clear-cut communication and computation phases
- Computation times even ignored in Distributed Computing community
- Communication times sometimes ignored in HPC community

Straightforward simulation in most cases

- Fill in a Gantt chart or count messages with a computer rather than by hand
- No need for a "simulation standard"

Large-Scale Distributed Systems Simulations?

Simple models justifiable at small scale

- Cluster computing (matrix multiply application on switched dedicated cluster)
- Small scale distributed systems

Hardly justifiable for Large-Scale Distributed Systems

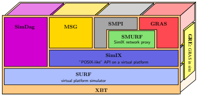
- Heterogeneity of components (hosts, links)
  - Quantitative: CPU clock, link bandwidth and latency
  - Qualitative: ethernet *vs* myrinet *vs* quadrics; Pentium *vs* Cell *vs* GPU
- Dynamicity
  - Quantitative: resource sharing $\leadsto$ availability variation
  - Qualitative: resource come and go (churn)
- Complexity
  - Hierarchical systems: grids of clusters of multi-processors being multi-cores
  - Resource sharing: network contention, QoS, batches
  - Multi-hop networks, non-negligible latencies
  - Middleware overhead (or optimizations)
  - Interference of computation and communication (and disk, memory, etc)

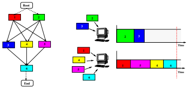
In silico approach to HPC experiments (simulation)

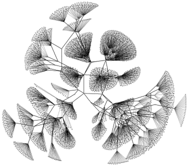
- Principle: Prototypes of applications, models of platforms
- Challenges: Get realistic results (experimental bias)

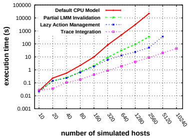
SimGrid: generic simulation framework for distributed applications

- Scalable (time and memory), modular, portable. +70 publications.
- Collaboration Loria / Inria Rhône-Alpes / CCIN2P3 / U. Hawaii









Other existing tools

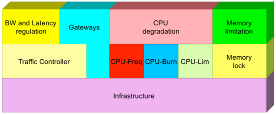
- *Large* amount of existing simulator for distributed platforms: GridSim, ChicSim, GES; P2PSim, PlanetSim, PeerSim; ns-2, GTNetS.
- Few are really usable: Diffusion, Software Quality Assurance, Long-term availability
- No other study the validity, the induced experimental bias

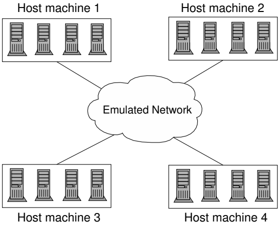
In vitro approach to HPC experiments (emulation)

- Principle: Injecting load on real systems for the experimental control
  ≈ *Slow platform down to put it in wanted experimental conditions*
- Challenges: Get realistic results, tool stack complex to deploy and use

Wrekavoc: applicative emulator

- Emulates CPU and network
- Homogeneous or Heterogeneous platforms





Nodes Virtualization

Other existing tools

- Network emulation: ModelNet, DummyNet, . . .
  Tools rather mature, but limited to network
- Applicative emulation: MicroGrid, eWan, Emulab
  Rarely (never?) used outside the lab where they were created

Existing evaluation tools for HPC ideas/applications

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
PlanetLab	virtualize	virtualize	virtualize	virtualize	access	uncontrolled	hundreds
ModelNet	-	-	emulation	emulation	lot material	controlled	dozens
MicroGrid	emulation	-	fine d.e.	emulation	none	controlled	hundreds
ns-2	-	-	fine d.e.	coarse d.e.	C++/tcl	controlled	<1,000
SSFNet	-	-	fine d.e.	coarse d.e.	Java	controlled	<100,000
GTNetS	-	-	fine d.e.	coarse d.e.	C++	controlled	<177,000
PlanetSim	-	-	cste time	coarse d.e.	Java	controlled	100,000
PeerSim	-	-	-	state machine	Java	controlled	1,000,000
ChicSim	coarse d.e.	-	coarse d.e.	coarse d.e.	C	controlled	thousands
OptorSim	coarse d.e.	amount	coarse d.e.	coarse d.e.	Java	controlled	few 100
GridSim	coarse d.e.	math	coarse d.e.	coarse d.e.	Java	controlled	few 1,000
SimGrid	math/d.e.	(some day)	math/d.e.	d.e./emul	C or Java	controlled	few 10,000

- Large platforms: getting access is problematic, fixed experimental settings
- Virtualization: no control over experimental settings
- Emulation: hard to setup, can have high overheads
- Packet-Level simulators: too network-centric (no CPU) and rather slow
- P2P simulators: great scalability, poor realism
- Grid simulators: limited scope, limited scalability, validity not assessed
- SimGrid: analytic network models ⇒ scalability and validity ok

Recap: Studying Large Distributed HPC Systems

Why? Compare aspects of the possible designs/algorithms/applications

- Response time
- Throughput
- Scalability
- Robustness
- Fault-tolerance
- Fairness

How? Several methodological approaches

- Theoretical approch: mathematical study [of algorithms]
  ☺ Better understanding, impossibility theorems; ☹ Everything NP-hard
- Experimentations (≈ in vivo): Real applications on Real platforms
  ☺ Believable; ☹ Hard and long. Experimental control? Reproducibility?
- Emulation (≈ in vitro): Real applications on Synthetic platforms
  ☺ Better experimental control; ☹ Even more difficult
- Simulation (in silico): Prototype of applications on model of systems
  ☺ Simple; ☹ Experimental bias

In Practice? A lot of tools exist; Some are even usable

- Key trade-off seem to be accuracy vs speed:
  The more abstract the fastest; The less abstract the most accurate. *Really?*

Simulation Validation: the FLASH example

FLASH project at Stanford

- Building large-scale shared-memory multiprocessors
- Went from conception, to design, to actual hardware (32-node)
- Used simulation heavily over 6 years

Authors compared simulation(s) to the real world

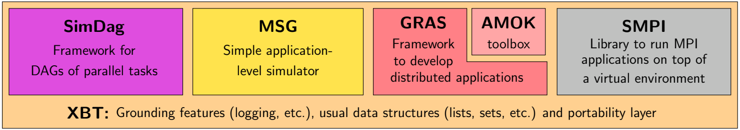
- Error is unavoidable (30% error in their case was not rare)
  Negating the impact of "we got 1.5% improvement"
- Complex simulators not ensuring better simulation results
  - Simple simulators worked better than sophisticated ones (which were unstable)
  - Simple simulators predicted trends as well as slower, sophisticated ones

  $\Rightarrow$ Should focus on simulating the important things
- Calibrating simulators on real-world settings is mandatory
- For FLASH, the simple simulator was all that was needed: Realistic $\approx$ Credible

Gibson, Kunz, Ofelt, Heinrich, *FLASH vs. (Simulated) FLASH: Closing the Simulation Loop*, Architectural Support for Programming Languages and Operating Systems, 2000

Outline

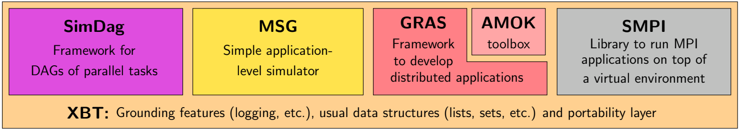
User-visible SimGrid Components



SimGrid user APIs

- SimDag: specify heuristics as DAG of (parallel) tasks
- MSG: specify heuristics as Concurrent Sequential Processes (Java/Ruby/Lua bindings available)
- GRAS: develop real applications, studied and debugged in simulator
- SMPI: simulate MPI codes

User-visible SimGrid Components



SimGrid user APIs

- SimDag: specify heuristics as DAG of (parallel) tasks
- MSG: specify heuristics as Concurrent Sequential Processes (Java/Ruby/Lua bindings available)
- GRAS: develop real applications, studied and debugged in simulator
- SMPI: simulate MPI codes

Which API should I choose?

- Your application is a DAG ⇝ SimDag
- You have a MPI code ⇝ SMPI
- You study concurrent processes, or distributed applications
  - You need graphs about several heuristics for a paper ⇝ MSG
  - You develop a real application (or want experiments on real platform) ⇝ GRAS
- Most popular API (for now): MSG

MSG: Heuristics for Concurrent Sequential Processes

(historical) Motivation

- Centralized scheduling does not scale
- SimDag (and its predecessor) not adapted to study decentralized heuristics
- MSG not strictly limited to scheduling, but particularly convenient for it

Main MSG abstractions

- **Agent:** some code, some private data, running on a given host
- **Task:** amount of work to do and of data to exchange
- Host: location on which agents execute
- Mailbox: similar to MPI tags

MSG: Heuristics for Concurrent Sequential Processes

(historical) Motivation

- Centralized scheduling does not scale
- SimDag (and its predecessor) not adapted to study decentralized heuristics
- MSG not strictly limited to scheduling, but particularly convenient for it

Main MSG abstractions

- **Agent:** some code, some private data, running on a given host
  set of functions + XML deployment file for arguments
- **Task:** amount of work to do and of data to exchange
  - `MSG_task_create(name, compute_duration, message_size, void *data)`
  - Communication: `MSG_task_{put,get}`, `MSG_task_Iprobe`
  - Execution: `MSG_task_execute`
    `MSG_process_sleep`, `MSG_process_{suspend,resume}`
- Host: location on which agents execute
- Mailbox: similar to MPI tags

SimGrid Usage Workflow: the MSG example (1/2)

1. Write the Code of your Agents

```
int master(int argc, char **argv) {

for (i = 0; i < number_of_tasks; i++) {
 t=MSG_task_create(name,comp_size,comm_size,data);
 sprintf(mailbox,"worker-%d",i % workers_count);
 MSG_task_send(t, mailbox);
}
```

```
int worker(int ,char**){

sprintf(my_mailbox,"worker-%d",my_id);
while(1) {
  MSG_task_receive(&task, my_mailbox);
  MSG_task_execute(task);
  MSG_task_destroy(task);
}
```

2. Describe your Experiment

XML Platform File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
<host name="host1" power="1E8"/>
<host name="host2" power="1E8"/>
...
<link name="link1" bandwidth="1E6"
                   latency="1E-2"  />
...
<route src="host1" dst="host2">
  <link:ctn id="link1"/>
</route>
</platform>
```

XML Deployment File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
<!-- The master process -->
<process host="host1" function="master">
 <argument value="10"/><!--argv[1]:#tasks-->
 <argument value="1"/><!--argv[2]:#workers-->
</process>

<!-- The workers -->
<process host="host2" function="worker">
  <argument value="0"/></process>
</platform>
```

SimGrid Usage Workflow: the MSG example (2/2)

3. Glue things together

```
int main(int argc, char *argv[ ]) {
  /* Bind agents' name to their function */
  MSG_function_register("master", &master);
  MSG_function_register("worker", &worker);

  MSG_create_environment("my_platform.xml");   /* Load a platform instance */
  MSG_launch_application("my_deployment.xml"); /* Load a deployment file */

  MSG_main(); /* Launch the simulation */

  INFO1("Simulation took %g seconds",MSG_get_clock());
}
```

4. Compile your code (linked against `-lsimgrid`), run it and enjoy

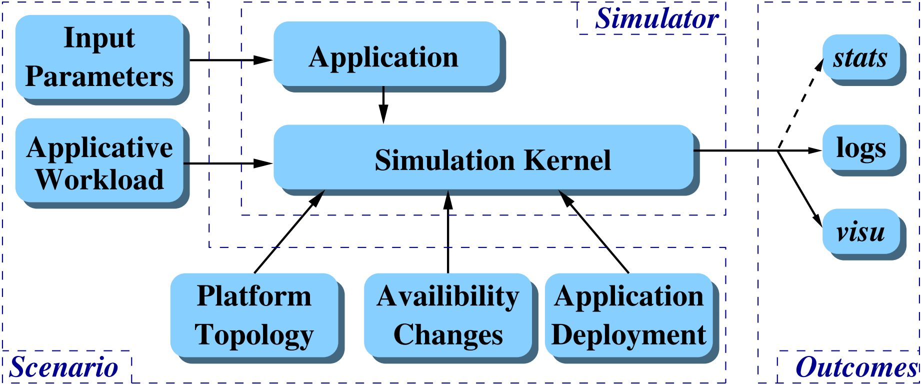
Executive summary, but representative

- Similar in others interfaces, but:
  - glue is generated by a script in GRAS and automatic in Java with introspection
  - in SimDag, no deployment file since no CSP
- Platform can contain trace informations, Higher level tags and Arbitrary data
- In MSG, applicative workload can also be externalized to a trace file

The MSG master/workers example: colorized output

```
$ ./my_simulator | MSG_visualization/colorize.pl
[      0.000] [  Tremblay:master    ] Got 3 workers and 6 tasks to process
[      0.000] [  Tremblay:master    ] Sending 'Task_0' to 'worker-0'
[      0.148] [  Tremblay:master    ] Sending 'Task_1' to 'worker-1'
[      0.148] [   Jupiter:worker    ] Processing 'Task_0'
[      0.347] [  Tremblay:master    ] Sending 'Task_2' to 'worker-2'
[      0.347] [    Fafard:worker    ] Processing 'Task_1'
[      0.476] [  Tremblay:master    ] Sending 'Task_3' to 'worker-0'
[      0.476] [   Ginette:worker    ] Processing 'Task_2'
[      0.803] [   Jupiter:worker    ] 'Task_0' done
[      0.951] [  Tremblay:master    ] Sending 'Task_4' to 'worker-1'
[      0.951] [   Jupiter:worker    ] Processing 'Task_3'
[      1.003] [    Fafard:worker    ] 'Task_1' done
[      1.202] [  Tremblay:master    ] Sending 'Task_5' to 'worker-2'
[      1.202] [    Fafard:worker    ] Processing 'Task_4'
[      1.507] [   Ginette:worker    ] 'Task_2' done
[      1.606] [   Jupiter:worker    ] 'Task_3' done
[      1.635] [  Tremblay:master    ] All tasks dispatched. Let's stop workers.
[      1.635] [   Ginette:worker    ] Processing 'Task_5'
[      1.637] [   Jupiter:worker    ] I'm done. See you!
[      1.857] [    Fafard:worker    ] 'Task_4' done
[      1.859] [    Fafard:worker    ] I'm done. See you!
[      2.666] [   Ginette:worker    ] 'Task_5' done
[      2.668] [  Tremblay:master    ] Goodbye now!
[      2.668] [   Ginette:worker    ] I'm done. See you!
[      2.668] [                     ] Simulation time 2.66766
```

SimGrid in a Nutshell



SimGrid is no simulator, but a simulation framework

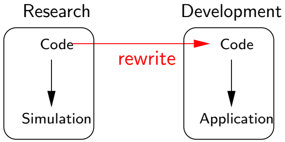
Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
  - Associated Tools
    - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
    - Visualization
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

Development of real distributed applications using a simulator

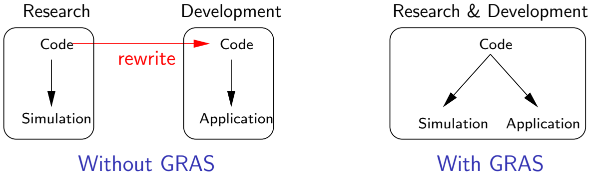


Without GRAS

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

Development of real distributed applications using a simulator

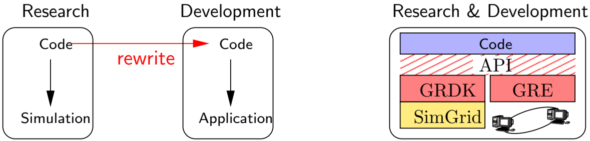


- Framework for Rapid Development of Distributed Infrastructure
  - Develop and tune on the simulator; Deploy *in situ* without modification

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

Development of real distributed applications using a simulator



Without GRAS

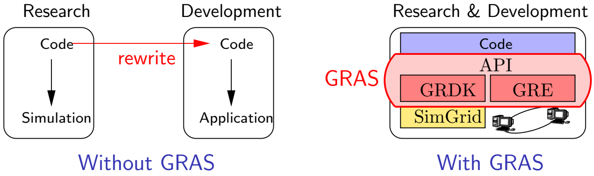
With GRAS

- Framework for Rapid Development of Distributed Infrastructure
  - Develop and tune on the simulator; Deploy *in situ* without modification
    How: One API, two implementations

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

Development of real distributed applications using a simulator

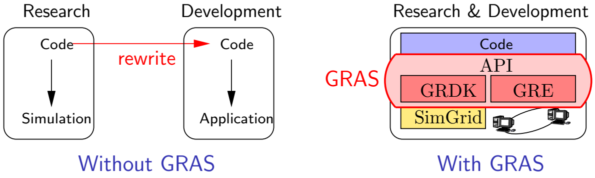


- Framework for Rapid Development of Distributed Infrastructure
  - Develop and tune on the simulator; Deploy *in situ* without modification
    How: One API, two implementations

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

Development of real distributed applications using a simulator



- Framework for Rapid Development of Distributed Infrastructure
  - Develop and tune on the simulator; Deploy *in situ* without modification
    How: One API, two implementations
- Efficient Grid Runtime Environment (result = application $\neq$ prototype)
  - Performance concern: efficient communication of structured data
    How: Efficient wire protocol (avoid data conversion)
  - Portability concern: because of grid heterogeneity
    How: ANSI C + autoconf + no dependency

Main concepts of the GRAS API

Agents (acting entities)

- Code (C function); Private data; Location (hosting computer)

Sockets (communication endpoints)

- Server socket: to receive messages
- Client socket: to contact a server (and receive answers)

Messages (what gets exchanged between agents)

- Semantic: Message type
- Payload described by data type description (fixed for a given type)
- Possible to attach automatic callbacks, or explicitely wait for messages

Differences with MSG

- Messages are typed (+callbacks), where MSG sends raw data chunks
- Socket oriented, where MSG uses mailboxes for rendez-vous
- Code can run in real settings too (so no over-simplification)

Exchanging structured data

GRAS wire protocol: NDR (Native Data Representation)

Avoid data conversion when possible:

- Sender writes data on socket as they are in memory
- If receiver's architecture does match, no conversion
- Receiver able to convert from any architecture

GRAS message payload can be any valid C type

- Structure, enumeration, array, pointer, ...
- Classical garbage collection algorithm to deep-copy it
- Cycles in pointed structures detected & recreated

Describing a data type to GRAS

Manual description (excerpt)

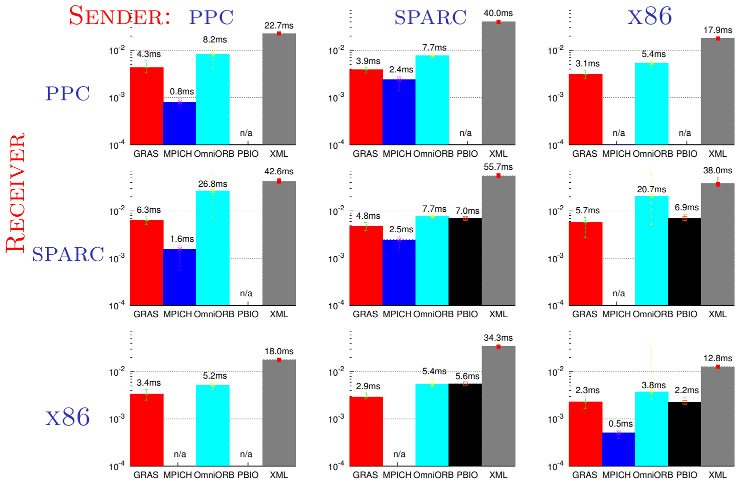
```
gras_datadesc_type_t gras_datadesc_struct(name);
gras_datadesc_struct_append(struct_type,name,field_type);
gras_datadesc_struct_close(struct_type);
```

Automatic description of vector

```
GRAS_DEFINE_TYPE(s_vect,
 struct s_vect {
  int cnt;
  double*data GRAS_ANNOTE(size,cnt);
 }
);
```

C declaration stored into a `char*` variable to be parsed at runtime

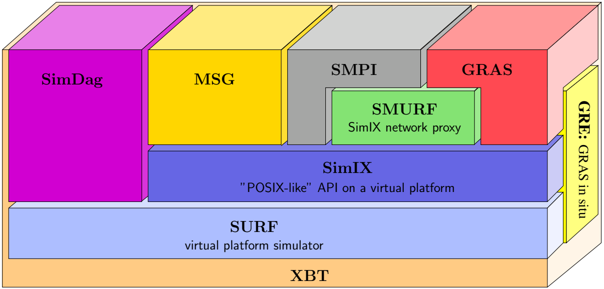
Communication Performance on a LAN

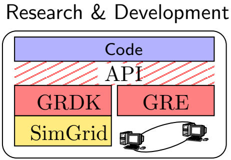


- MPICH twice as fast as GRAS, but cannot mix little- and big-endian Linux
- PBIO broken on PPC
- XML much slower (extra conversions + verbose wire encoding)

GRAS is the better compromise between performance and portability

GRAS eases infrastructure development





With GRAS

GRDK: Grid Research & Development Kit

- API for (explicitly) distributed applications
- Study applications in the comfort of the simulator

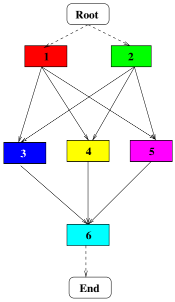
GRE: Grid Runtime Environment

- Efficient: twice as slow as MPICH, faster than OmniORB, PBIO, XML
- Portable: Linux (11 CPU archs); *Windows*; Mac OS X; Solaris; IRIX; AIX
- Simple and convenient:
  - API simpler than classical communication libraries (automatic IDL)
  - Easy to deploy: C ANSI; no dependency; $<$400kb

Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
  - Associated Tools
    - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
    - Visualization
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

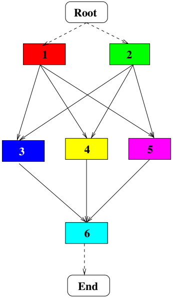
SimDag: Comparing Scheduling Heuristics for DAGs

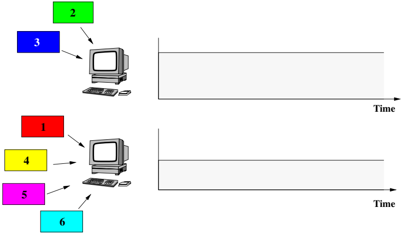


Main functionalities

1. Create a DAG of tasks
   - Vertices: tasks (either communication or computation)
   - Edges: precedence relation

SimDag: Comparing Scheduling Heuristics for DAGs

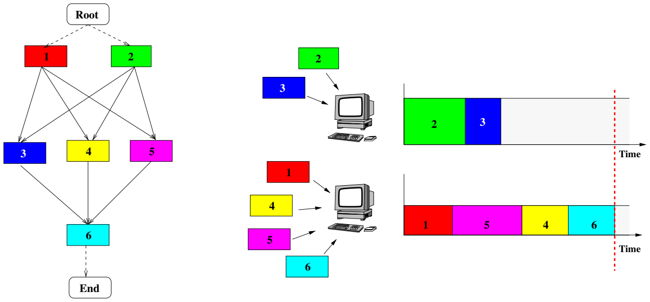




Main functionalities

1. Create a DAG of tasks
   - Vertices: tasks (either communication or computation)
   - Edges: precedence relation
2. Schedule tasks on resources

SimDag: Comparing Scheduling Heuristics for DAGs



Main functionalities

1. Create a DAG of tasks
   - Vertices: tasks (either communication or computation)
   - Edges: precedence relation
2. Schedule tasks on resources
3. Run the simulation (respecting precedences)
   $\rightsquigarrow$ Compute the makespan

The SimDag interface

DAG creation

- Creating tasks: `SD_task_create(name, data)`
- Creating dependencies: `SD_task_dependency_{add/remove}(src,dst)`

Scheduling tasks

- `SD_task_schedule(task, workstation_number, *workstation_list, double *comp_amount, double *comm_amount, double rate)`
  - Tasks are parallel by default; simply put workstation_number to 1 if not
  - Communications are regular tasks, `comm_amount` is a matrix
  - Both computation and communication in same task possible
  - `rate`: To slow down non-CPU (resp. non-network) bound applications
- `SD_task_unschedule`, `SD_task_get_start_time`

Running the simulation

- `SD_simulate(double how_long)` (how_long $< 0 \leadsto$ until the end)
- `SD_task_{watch/unwatch}`: simulation stops as soon as task's state changes

Full API in the doxygen-generated documentation

SMPI: Running MPI applications on top of SimGrid

Motivations

- Reproducible experimentation of MPI code (debugging)
- Test MPI code on still-to-build platform (dimensioning)

How it works

- `smpicc` changes MPI calls into SMPI ones (gettimeofday also intercepted)
- `smpirun` starts a classical simulation obeying `-hostfile` and `-np`

⇒ Runs unmodified MPI code after recompilation

Implemented calls

- Isend; Irecv. Recv; Send; Sendrecv. Wait; Waitall; Waitany. Reduce; Allreduce.
- Barrier; Bcast; Reduce; Allreduce (cmd line switch between binary or flat tree)
- Comm_size; Comm_rank; Comm_split. Wtime. Init; Finalize; Abort.

Current Work

- Implement the rest of the API; Test it more througfully
- Use it to validate SimGrid at application level (with NAS *et Al.*)

Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
  - Associated Tools
    - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
    - Visualization
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

Analytic Models underlying the SimGrid Framework

Main challenges for SimGrid design

- Simulation accuracy:
  - Designed for HPC scheduling community $\rightsquigarrow$ don't mess with the makespan!
  - At the very least, understand validity range
- Simulation speed:
  - Users conduct large parameter-sweep experiments over alternatives

Microscopic simulator design

- Simulate the packet movements and routers algorithms
- Simulate the CPU actions (or micro-benchmark classical basic operations)
- Hopefully very accurate, but *very* slow (simulation time $\gg$ simulated time)

Going faster while remaining reasonable?

- Need to come up with macroscopic models for each kind of resource
- Main issue: resource sharing. *Emerge* naturally in microscopic approach:
  - Packets of different connections interleaved by routers
  - CPU cycles of different processes get slices of the CPU

Modeling a Single Resource

Basic model: $Time = L + \frac{size}{B}$

- Resource work at given *rate* (B, in MFlop/s or Mb/s)
- Each use have a given latency (L, in s)

Modeling CPU

- Resource delivers *pow* flop / sec; task require *size* flop $\Rightarrow$ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

Modeling Single-Hop Networks

- Simplistic: $T = \lambda + \frac{size}{\beta}$;
- More accurate: [Padhye, Firoiu, Towsley, Krusoe 2000]

$$B = min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \times min(1, 3\sqrt{3bp/8}) \times p(1+32p^2)}\right)$$

- p: loss indication rate
- b: #packages acknowledged per ACK
- T_0: TCP average retransmission timeout value

- Let's keep instanciable: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

Modeling a Single Resource

Basic model: $Time = L + \frac{size}{B}$

- Resource work at given *rate* (B, in MFlop/s or Mb/s)
- Each use have a given latency (L, in s)

Modeling CPU

- Resource delivers *pow* flop / sec; task require *size* flop $\Rightarrow$ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

Modeling Single-Hop Networks

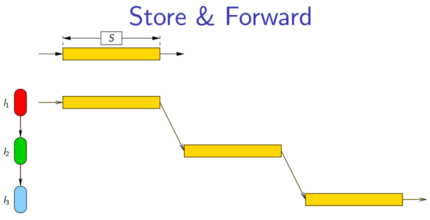
- Simplistic: $T = \lambda + \frac{size}{\beta}$;
- More accurate: [Padhye, Firoiu, Towsley, Krusoe 2000]

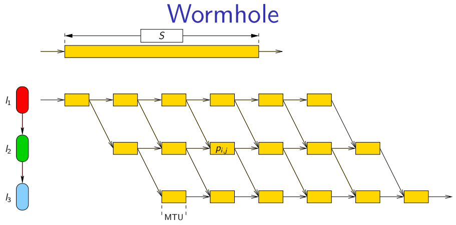
$$B = min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \times min(1, 3\sqrt{3bp/8}) \times p(1+32p^2)}\right)$$

  - p: loss indication rate
  - b: #packages acknowledged per ACK
  - T_0: TCP average retransmission timeout value
  - p and b not known in general (model hard to instanciable)
- Let's keep instanciable: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

Modeling Multi-Hop Networks

Simplistic Models





- Quite natural:
  cf. time to go from city to city
- Plainly Wrong:
  Data not stored on each router

- Appealing: (& used in most tools)
  Remember your networking class?
- Really inaccurate:
  IP fragmentation, TCP Congestion

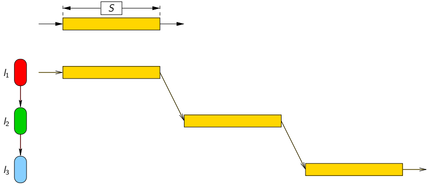
NS2 and other packet-level

- study the path of each and every network packet
- ☺ Realism commonly accepted; ☹ Sloooooow

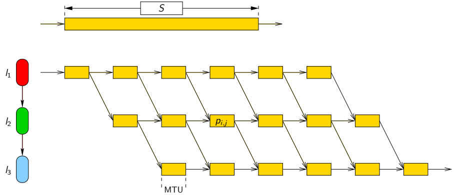
Modeling Multi-Hop Networks

Simplistic Models

Store & Forward



Wormhole



- Quite natural:
  cf. time to go from city to city
- Plainly Wrong:
  Data not stored on each router

- Appealing: (& used in most tools)
  Remember your networking class?
- Really inaccurate:
  IP fragmentation, TCP Congestion

What's in between these two approaches?

NS2 and other packet-level

- study the path of each and every network packet
- ☺ Realism commonly accepted; ☹ Sloooooow

Analytical Network Models

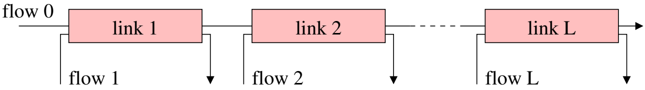
TCP bandwidth sharing studied by several authors

- Data streams modeled as fluids in pipes
- Same model for single stream/multiple links or multiple stream/multiple links

Analytical Network Models

TCP bandwidth sharing studied by several authors

- Data streams modeled as fluids in pipes
- Same model for single stream/multiple links or multiple stream/multiple links



Notations

- $\mathcal{L}$: set of links
- C_l: capacity of link l ($C_l > 0$)
- n_l: amount of flows using link l
- $\mathcal{F}$: set of flows; $f \in P(\mathcal{L})$
- λ_f : transfer rate of f

Feasibility constraint

- Links deliver their capacity at most: $\forall l \in \mathcal{L}, \sum_{f \ni l} \lambda_f \leq C_l$

Max-Min Fairness

Objective function: maximize $\min_{f \in \mathcal{F}}(\lambda_f)$

- Equilibrium reached if increasing any λ_f decreases a λ'_f (with $\lambda_f > \lambda'_f$)
- Very reasonable goal: gives fair share to anyone
- Optionally, one can add prorities w_i for each flow i
  $\leadsto$ maximizing $\min_{f \in \mathcal{F}}(w_f \lambda_f)$

Bottleneck links

- For each flow f, one of the links is the limiting one l
  (with more on that link l, the flow f would get more overall)
- The objective function gives that l is saturated, and f gets the biggest share

$$\forall f \in \mathcal{F}, \exists l \in f, \quad \sum_{f' \ni l} \lambda_{f'} = C_l \quad \text{and} \quad \lambda_f = \max\{\lambda_{f'}, f' \ni l\}$$

L. Massoulié and J. Roberts, *Bandwidth sharing: objectives and algorithms*,
IEEE/ACM Trans. Netw., vol. 10, no. 3, pp. 320-328, 2002.

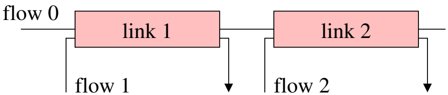
Max-Min Fairness on Homogeneous Linear Network

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

Algorithm: loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_i}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network



$C_1 = C \quad n_1 = 2$

$C_2 = C \quad n_2 = 2$

$\lambda_0 =$

$\lambda_1 =$

$\lambda_2 =$

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

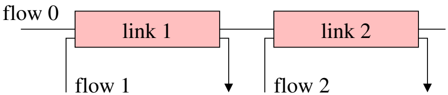
Max-Min Fairness on Homogeneous Linear Network

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

Algorithm: loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_i}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network



$$C_1 = C \qquad n_1 = 2$$
$$C_2 = C \qquad n_2 = 2$$

$$\lambda_0 = C/2$$
$$\lambda_1 = C/2$$
$$\lambda_2 =$$

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

$\Rightarrow$ $\lambda_0 = C/2$ and $\lambda_1 = C/2$

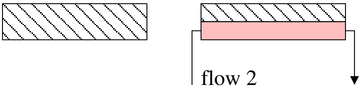
Max-Min Fairness on Homogeneous Linear Network

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

Algorithm: loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_i}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network



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

$$\lambda_0 = C/2$$
$$\lambda_1 = C/2$$
$$\lambda_2 =$$

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

$\Rightarrow$ $\lambda_0 = C/2$ and $\lambda_1 = C/2$

- Remove flows 0 and 1; Update links' capacity

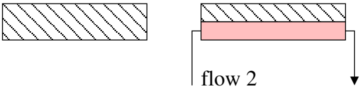
Max-Min Fairness on Homogeneous Linear Network

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

Algorithm: loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_i}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network



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

$\lambda_0 = C/2$
$\lambda_1 = C/2$
$\lambda_2 = C/2$

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

$\Rightarrow$ $\lambda_0 = C/2$ and $\lambda_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\lambda_1 = C/2$

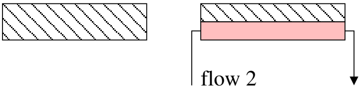
Max-Min Fairness on Homogeneous Linear Network

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

Algorithm: loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_i}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network



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

$\lambda_0 = C/2$
$\lambda_1 = C/2$
$\lambda_2 = C/2$

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

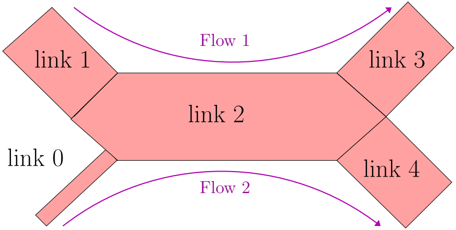
$\Rightarrow$ $\lambda_0 = C/2$ and $\lambda_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\lambda_1 = C/2$

We're done computing the bandwidth allocated to each flow

Max-Min Fairness Computation: Backbone Example

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.



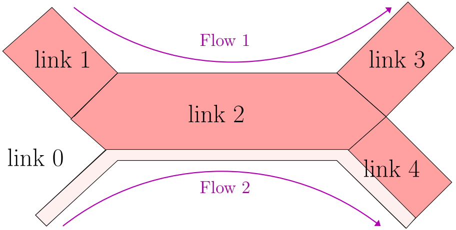
$C_0 = 1$ $n_0 = 1$
$C_1 = 1000$ $n_1 = 1$
$C_2 = 1000$ $n_2 = 2$
$C_3 = 1000$ $n_3 = 1$
$C_4 = 1000$ $n_4 = 1$

$\lambda_1 =$
$\lambda_2 =$

- The limiting link is 0

Max-Min Fairness Computation: Backbone Example

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.



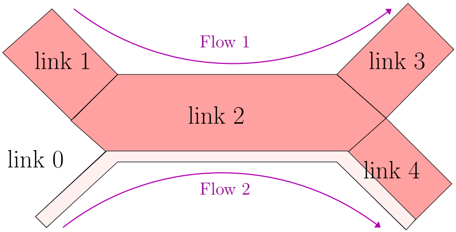
$C_0 = 0$ $n_0 = 0$
$C_1 = 1000$ $n_1 = 1$
$C_2 = 999$ $n_2 = 1$
$C_3 = 1000$ $n_3 = 1$
$C_4 = 999$ $n_4 = 0$

$\lambda_1 =$
$\lambda_2 = 1$

- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links

Max-Min Fairness Computation: Backbone Example

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.



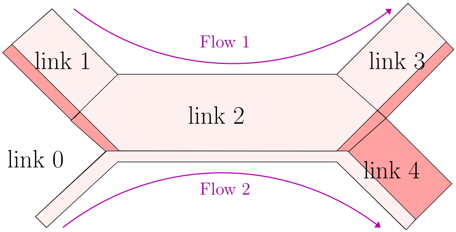
$C_0 = 0$ $n_0 = 0$
$C_1 = 1000$ $n_1 = 1$
$C_2 = 999$ $n_2 = 1$
$C_3 = 1000$ $n_3 = 1$
$C_4 = 999$ $n_4 = 0$

$\lambda_1 =$
$\lambda_2 = 1$

- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links
- The limiting link is 2

Max-Min Fairness Computation: Backbone Example

C_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.

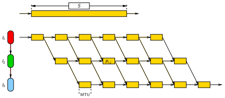


$C_0 = 0$ $n_0 = 0$
$C_1 = 1$ $n_1 = 0$
$C_2 = 0$ $n_2 = 0$
$C_3 = 1$ $n_3 = 0$
$C_4 = 999$ $n_4 = 0$

$\lambda_1 = 999$
$\lambda_2 = 1$

- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links
- The limiting link is 2
- This fixes $\lambda_1 = 999$

Side note: OptorSim 2.1 on Backbone



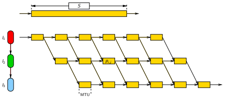
OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$

Side note: OptorSim 2.1 on Backbone

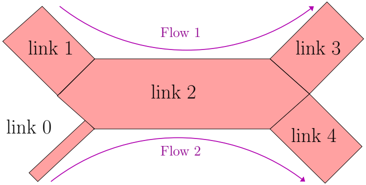


OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$

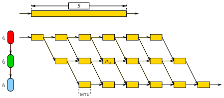


$C_0 = 1$	$n_1 = 1$	share =
$C_1 = 1000$	$n_1 = 1$	share =
$C_2 = 1000$	$n_2 = 2$	share =
$C_3 = 1000$	$n_3 = 1$	share =
$C_4 = 1000$	$n_4 = 1$	share =

$\lambda_1 =$

$\lambda_2 =$

Side note: OptorSim 2.1 on Backbone

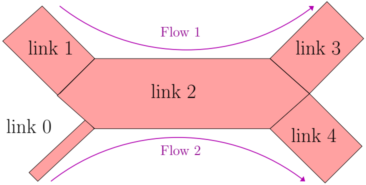


OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using wormhole

Unfortunately, "strange" resource sharing:

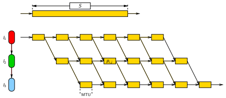
1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$C_0 = 1$	$n_1 = 1$	share $= 1$
$C_1 = 1000$	$n_1 = 1$	share $= 1000$
$C_2 = 1000$	$n_2 = 2$	share $= 500$
$C_3 = 1000$	$n_3 = 1$	share $= 1000$
$C_4 = 1000$	$n_4 = 1$	share $= 1000$

$\lambda_1 = min(1000, 500, 1000)$
$\lambda_2 = min(\ 1 \quad , 500, 1000)$

Side note: OptorSim 2.1 on Backbone

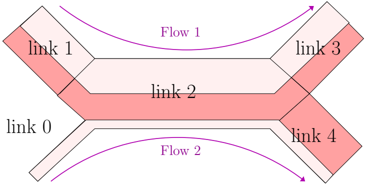


OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using wormhole

Unfortunately, "strange" resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$\begin{array}{lll} C_0 = 1 & n_1 = 1 & \text{share} = 1 \\ C_1 = 1000 & n_1 = 1 & \text{share} = 1000 \\ C_2 = 1000 & n_2 = 2 & \text{share} = \mathit{500} \\ C_3 = 1000 & n_3 = 1 & \text{share} = 1000 \\ C_4 = 1000 & n_4 = 1 & \text{share} = 1000 \end{array}$$

$$\lambda_1 = min(1000, 500, 1000) = \mathbf{500}!!$$
$$\lambda_2 = min(\ 1\ \ , 500, 1000) = 1$$

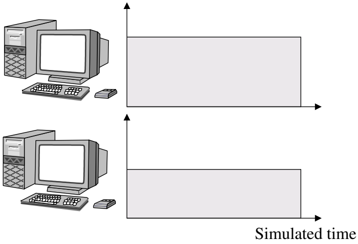
λ_1 limited by link 2, but 499 still unused on link 2

This "unwanted feature" is even listed in the README file...

How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

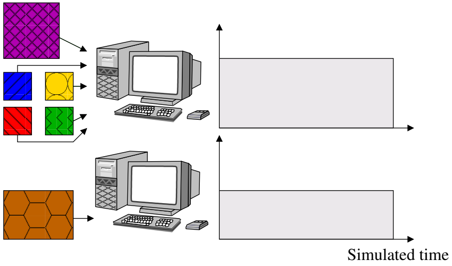


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

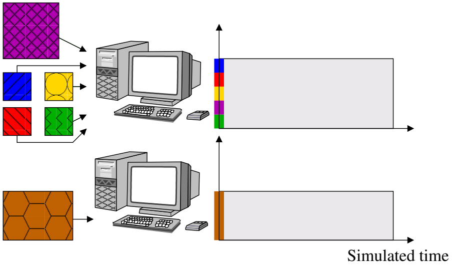


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

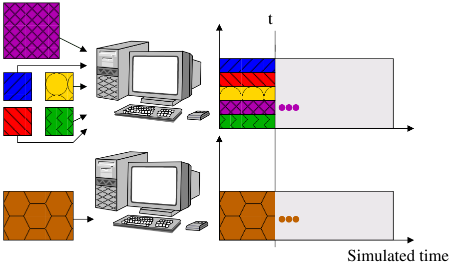


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

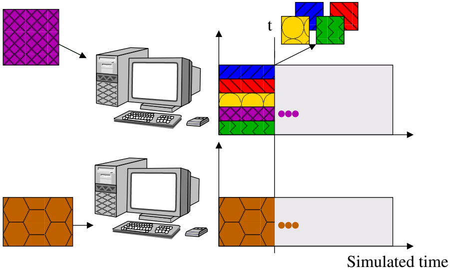


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

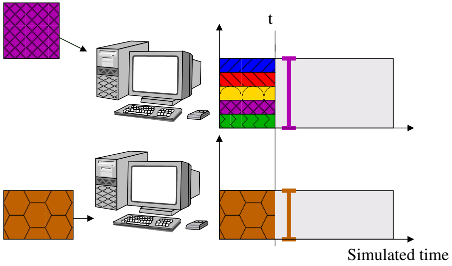


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

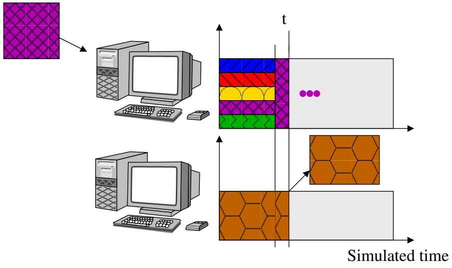


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

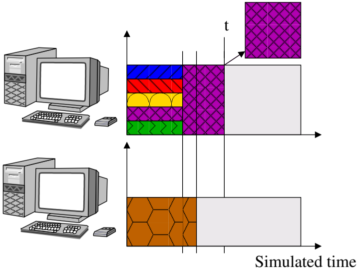


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

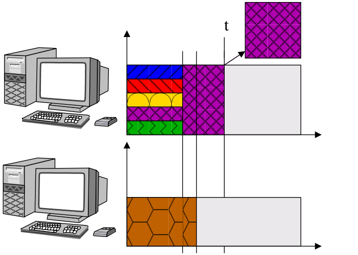


How are these models used in practice?

Simulation kernel main loop

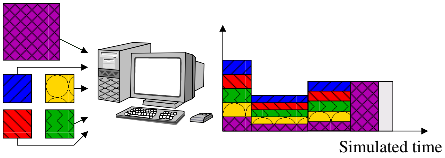
Data: set of resources with working rate

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



Availability traces are just events

$t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*



Also qualitative state changes (on/off)

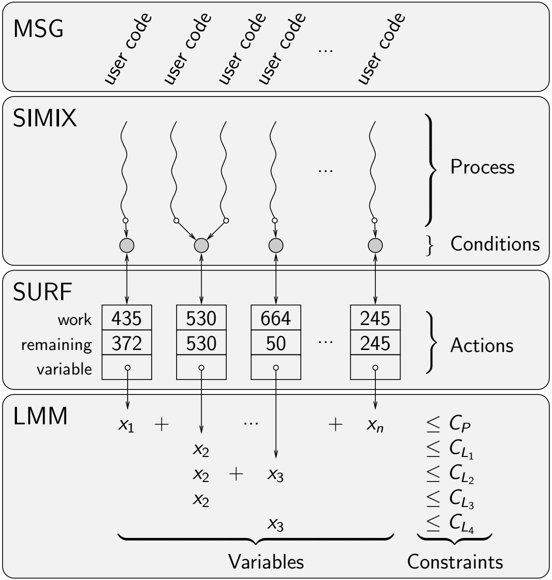
SimGrid Internals in a Nutshell for Users

SimGrid Layers

- MSG: User interface
- Simix: processes, synchro
- SURF: Resources
- (LMM: MaxMin systems)

Changing the Model

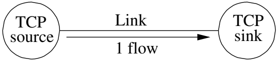
- "--cfg=network_model"
- Several fluid models
- Several constant time
- GTNetS wrapper
- Build your own (!)



Outline

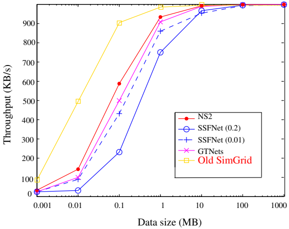
Validation experiments on a single link

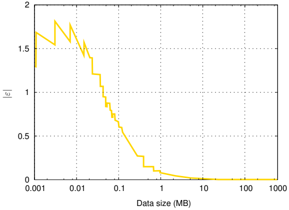
Experimental settings



- Compute achieved bandwidth as function of S
- Fixed L=10ms and B=100MB/s

Evaluation Results



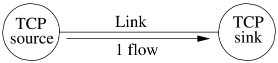


- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default
- GTNetS is equally distant from others

- Old SimGrid model omitted slow start effects

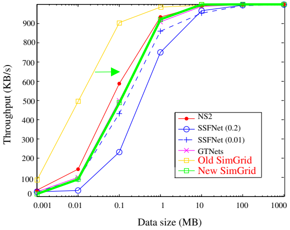
Validation experiments on a single link

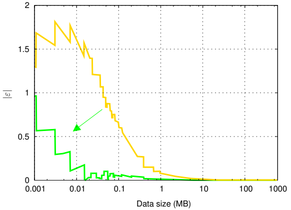
Experimental settings



- Compute achieved bandwidth as function of S
- Fixed L=10ms and B=100MB/s

Evaluation Results





- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default
- GTNetS is equally distant from others

- Old SimGrid model omitted slow start effects

$\Rightarrow$ Statistical analysis of GTNetS slow-start

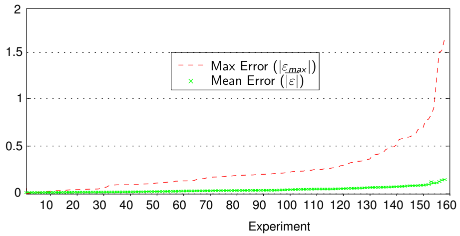
$\rightsquigarrow$ Better instantiation of MaxMin model
$\beta'' \rightsquigarrow .92 \times \beta'$; $\lambda \rightsquigarrow 10.4 \times \lambda$

- Resulting validity range quite acceptable

S	$\overline{\lvert\varepsilon\rvert}$	$\lvert\varepsilon_{max}\rvert$
$S < 100KB$	$\approx 12\%$	$\approx 162\%$
$S > 100KB$	$\approx 1\%$	$\approx 6\%$

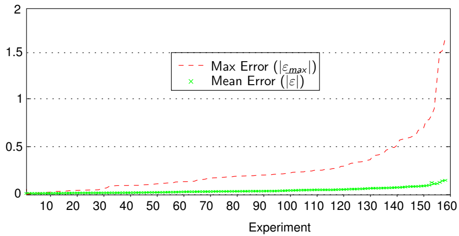
Validation experiments on random platforms

- 160 Platforms (generator: BRITE)
- $\beta \in [10{,}128]$ MB/s; $\lambda \in [0; 5]$ ms
- Flow size: S=10MB
- #flows: 150; #nodes $\in [50; 200]$
- $\overline{|\varepsilon|} < 0.2$ (*i.e.*, $\approx$ 22%);
  $|\varepsilon_{max}|$ still challenging up to 461%



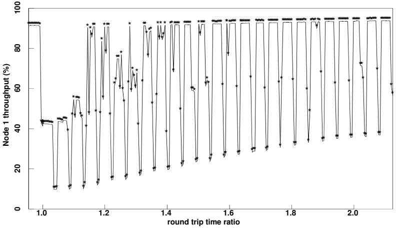
Validation experiments on random platforms

- 160 Platforms (generator: BRITE)
- $\beta \in [10{,}128]$ MB/s; $\lambda \in [0;5]$ ms
- Flow size: S=10MB
- #flows: 150; #nodes $\in [50;200]$
- $\overline{|\varepsilon|} < 0.2$ (*i.e.*, $\approx$ 22%);
  $|\varepsilon_{max}|$ still challenging up to 461%



Maybe the error is not SimGrid's

- Big error because GTNetS multi-phased
- Seen the same in NS3, emulation, ...
- Phase Effect: Periodic and deterministic traffic may resonate [Floyd&Jacobson 91]
- Impossible in Internet (thx random noise)



⇝ We're adding random jitter to continue SimGrid validation

Simulation scalability assessment (how big?)

Master/Workers on amd64 with 4Gb

#tasks	Context mecanism	#Workers 100	500	1,000	5,000	10,000	25,000
1,000	ucontext	0.16	0.19	0.21	0.42	0.74	1.66
	pthread	0.15	0.18	0.19	0.35	0.55	⋆
	java	0.41	0.59	0.94	7.6	27.	⋆
10,000	ucontext	0.48	0.52	0.54	0.83	1.1	1.97
	pthread	0.51	0.56	0.57	0.78	0.95	⋆
	java	1.6	1.9	2.38	13.	40.	⋆
100,000	ucontext	3.7	3.8	4.0	4.4	4.5	5.5
	pthread	4.7	4.4	4.6	5.0	5.23	⋆
	java	14.	13.	15.	29.	77.	⋆
1,000,000	ucontext	36.	37.	38.	41.	40.	41.
	pthread	42.	44.	46.	48.	47.	⋆
	java	121.	130.	134.	163.	200.	⋆

⋆: #semaphores reached system limit (2 semaphores per user process, System limit = 32k semaphores)

- These results are old already
- v3.3.3 is 30% faster
- v3.3.4 $\rightsquigarrow$ lazy evaluation

Extensibility with UNIX contextes

#tasks	Stack size	#Workers 25,000	50,000	100,000	200,000
1,000	128Kb	1.6	†	†	†
	12Kb	0.5	0.9	1.7	3.2
10,000	128Kb	2	†	†	†
	12Kb	0.8	1.2	2	3.5
100,000	128Kb	5.5	†	†	†
	12Kb	3.7	4.1	4.8	6.7
1,000,000	128Kb	41	†	†	†
	12Kb	33	33.6	33.7	35.5
5,000,000	128Kb	206	†	†	†
	12Kb	161	167	161	165

Scalability limit of GridSim

- 1 user process = 3 java threads (code, input, output)
- System limit = 32k threads

⇒ at most 10,922 user processes

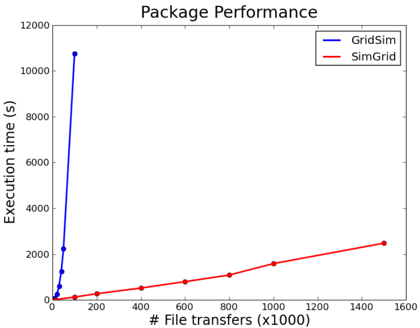
†: out of memory

Simulation scalability assessment (how fast?)

During Summer 2009, 2 interns @CERN evaluated grid simulators

- Attempted to simulate one day on their data grid (1.5 million file transfers)
- Their final requirements:
  - Basic processing induce 30M operations daily
  - User requests induce ≈2M operations daily
  - Evaluations should consider one month of operation

Findings

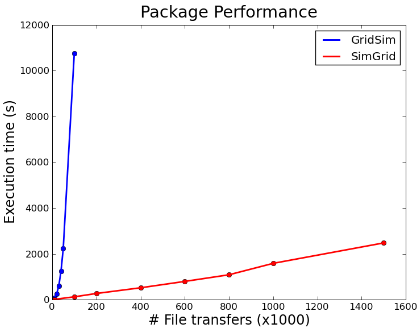


Simulation scalability assessment (how fast?)

During Summer 2009, 2 interns @CERN evaluated grid simulators

- Attempted to simulate one day on their data grid (1.5 million file transfers)
- Their final requirements:
  - Basic processing induce 30M operations daily
  - User requests induce ≈2M operations daily
  - Evaluations should consider one month of operation

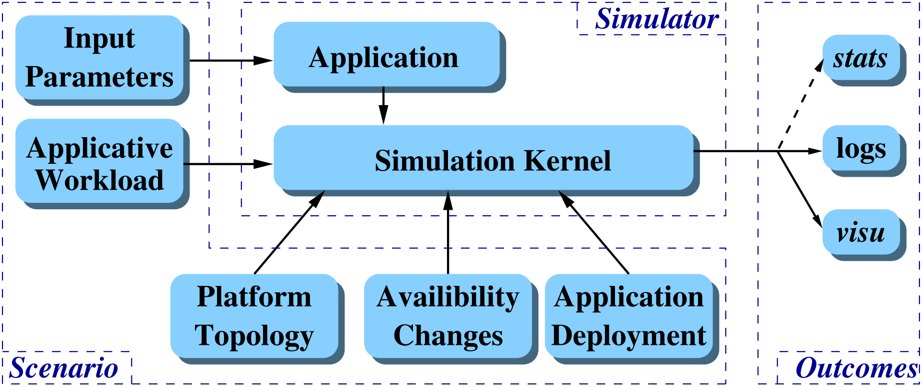
Findings



Experiment to be redone?

- Controlled experimental settings
- With recent versions of the tools
- More metrics
- Better if not done by us (you?)

SimGrid Workflow



Simulation is only one piece of the workflow

- Needed Input:
  - Platform (quantitative and qualitative)
  - Application (code and deployment; workload)
- Provided Output: Text logs, Graphical Visualization

Platform Instantiation

To use a simulator, one must instantiate the models

Key questions

- How can I run my tests on realistic platforms? What is a realistic platform?
- What are platform parameters? What are their values in real platforms?

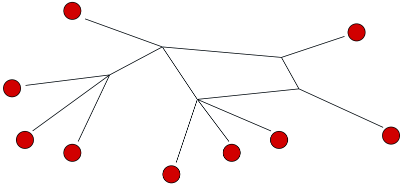
Sources of platform descriptions

- Manual modeling: define the characteristics with your sysadmins
- Synthetic platform generator: use random generators
- Automatic mapping: automated tomography tool

What is a Platform Instance Anyway?

Structural description

- Hosts list
- Links and interconnexion topology



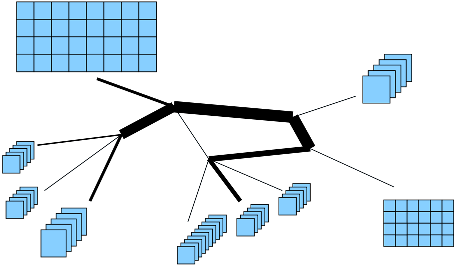
What is a Platform Instance Anyway?

Structural description

- Hosts list
- Links and interconnexion topology

Peak Performance

- Bandwidth and Latencies
- Processing capacity



What is a Platform Instance Anyway?

Structural description

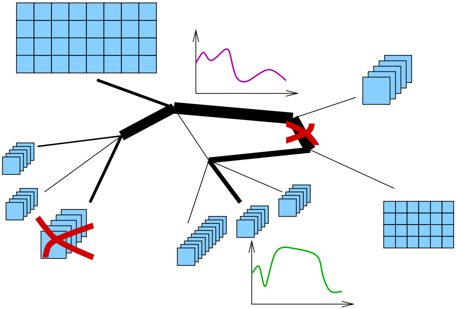
- Hosts list
- Links and interconnexion topology

Peak Performance

- Bandwidth and Latencies
- Processing capacity

Background Conditions

- Load
- Failures



Platform description for SimGrid

Example of XML file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
  <host id="Jacquelin" power="137333000"/>
  <host id="Boivin" power="98095000"/>


  <link id="1" bandwidth="3430125" latency="0.000536941"/>
  <route src="Jacquelin" dst="Boivin"><link:ctn id="1"/></route>
  <route src="Boivin" dst="Jacquelin"><link:ctn id="1"/></route>
</platform>
```

- Declare all your hosts, with their computing power
- Declare all your links, with bandwidth and latency
- Declare routes from each host to each host (list of links)

Platform description for SimGrid

Example of XML file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
  <host id="Jacquelin" power="137333000"/>
  <host id="Boivin" power="98095000"/>


  <link id="1" bandwidth="3430125" latency="0.000536941"/>
  <route src="Jacquelin" dst="Boivin"><link:ctn id="1"/></route>
  <route src="Boivin" dst="Jacquelin"><link:ctn id="1"/></route>
</platform>
```

- Declare all your hosts, with their computing power
  other attributes:
  - availability_file: trace file to let the power vary
  - state_file: trace file to specify whether the host is up/down
- Declare all your links, with bandwidth and latency
  - bandwidth_file, latency_file, state_file: trace files
  - sharing_policy $\in \{\text{shared}, \text{fatpipe}\}$ (fatpipe $\rightsquigarrow$ no sharing)
- Declare routes from each host to each host (list of links)

Platform description for SimGrid

Example of XML file

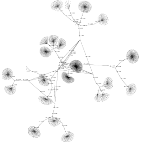
```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
  <host id="Jacquelin" power="137333000"/>
  <host id="Boivin" power="98095000">
    <prop key="someproperty" value="somevalue"/> <!-- attach arbitrary data to hosts/links -->
  </host>
  <link id="1" bandwidth="3430125" latency="0.000536941"/>
  <route src="Jacquelin" dst="Boivin"><link:ctn id="1"/></route>
  <route src="Boivin" dst="Jacquelin"><link:ctn id="1"/></route>
</platform>
```

- Declare all your hosts, with their computing power
  other attributes:
  - availability_file: trace file to let the power vary
  - state_file: trace file to specify whether the host is up/down
- Declare all your links, with bandwidth and latency
  - bandwidth_file, latency_file, state_file: trace files
  - sharing_policy $\in$ {shared, fatpipe} (fatpipe $\leadsto$ no sharing)
- Declare routes from each host to each host (list of links)
- Arbitrary data can be attached to components using the <prop> tag

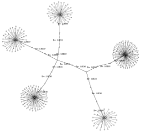
Platform Catalog

Several Existing Platforms Modeled

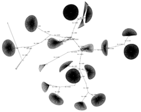
Grid'5000
9 sites, 25 clusters
1,528 hosts



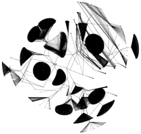
DAS 3
5 clusters
277 hosts



GridPP
18 clusters
7,948 hosts



LCG
113 clusters
44,184 hosts



Files available from the Platform Description Archive
`http://pda.gforge.inria.fr`

(+ tool to extract platform subsets)

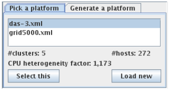
Synthetic Topology Generation

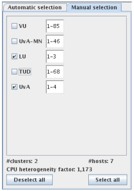
Designing a Realistic Platform Generator

- Examine real platforms; Discover principles; Implement a generator
- Subject of studies in Networking for years ⇒ Loads of generation methods

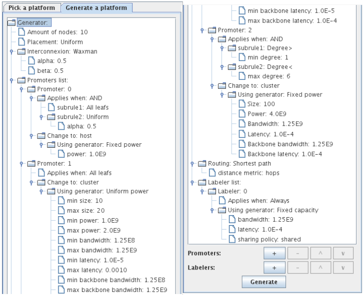
Simulacrum: Generic GUI to generate SimGrid platform files

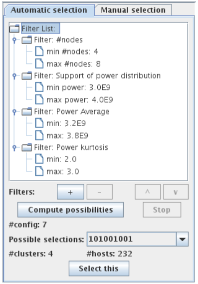
Selecting from existing





Generating + Filtering





Other tools

- Several well known generators for networking community, eg Brite
- Grid-G: All in one framework, Grid specific

Automatic Network Mapping

Main Issue of synthetic generators: Realism!

- Solution: Actually map a real platform

Automatic Network Mapping

Main Issue of synthetic generators: Realism!

- Solution: Actually map a real platform
- Tomography: 2-steps process (end-to-end Measurements; Reconstruct a graph)

Several levels of information (depending on the OSI layer)

- Physical inter-connexion map (wires in the walls)
- Routing infrastructure (path of network packets, from router to switch)
- Application level (focus on effects – bandwidth & latency – not causes)

Automatic Network Mapping

Main Issue of synthetic generators: Realism!

- Solution: Actually map a real platform
- Tomography: 2-steps process (end-to-end Measurements; Reconstruct a graph)

Several levels of information (depending on the OSI layer)

- Physical inter-connexion map (wires in the walls)
- Routing infrastructure (path of network packets, from router to switch)
- Application level (focus on effects – bandwidth & latency – not causes)

  Our goal: conduct experiments at application level, not administrating tool

The ALNeM project (Application-Level Network Mapper)

- Long-term goal: be a tool providing topology to network-aware applications
- Short-term goal: allow the study of network mapping algorithms
- Project started in 2002, still underway ☹

Automatic Network Mapping

Main Issue of synthetic generators: Realism!

- Solution: Actually map a real platform
- Tomography: 2-steps process (end-to-end Measurements; Reconstruct a graph)

Several levels of information (depending on the OSI layer)

- Physical inter-connexion map (wires in the walls)
- Routing infrastructure (path of network packets, from router to switch)
- Application level (focus on effects – bandwidth & latency – not causes)

  Our goal: conduct experiments at application level, not administrating tool

The ALNeM project (Application-Level Network Mapper)

- Long-term goal: be a tool providing topology to network-aware applications
- Short-term goal: allow the study of network mapping algorithms
- Project started in 2002, still underway ☹

Measurement step

- Network level tools (BGP, SNMP, ICMP)
  - use restricted for security reason
  - hard to get a App-Level view from them
- We rely on simple E2E measurements (latency/bandwidth)

Evaluation methodology

How to evaluate the reconstruction algorithms' quality?

Several evaluation metrics

1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:

$$Interf\left((a,b),(c,d)\right) = 1 \text{ iff } \frac{BW(a \to b)}{BW(a \to b \parallel c \to d)} \approx 2$$

3. Compare application running times (application-level)

	Comm. schema	// comm	# steps
Token-ring	Ring	No	1
Broadcast	Tree	No	1
All2All	Clique	Yes	1
Parallel Matrix Multiplication	2D	Yes	$\sqrt{procs}$

Evaluation methodology

How to evaluate the reconstruction algorithms' quality?

Several evaluation metrics

1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:

$$Interf\left((a,b),(c,d)\right) = 1 \text{ iff } \frac{BW(a \to b)}{BW(a \to b \parallel c \to d)} \approx 2$$

3. Compare application running times (application-level)

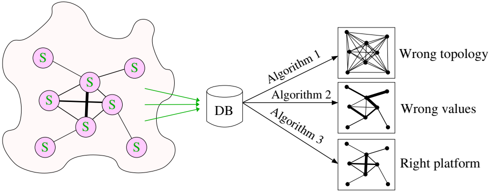
	Comm. schema	// comm	# steps
Token-ring	Ring	No	1
Broadcast	Tree	No	1
All2All	Clique	Yes	1
Parallel Matrix Multiplication	2D	Yes	$\sqrt{procs}$

(other) Methodological Challenge

- Goal: Quantify similarity between initial and reconstructed platforms
- Impossible to test against real platform Reconstructed platform doesn't exist
- Testing on simulator: both initial and reconstructed platforms are simulated
  Leveraging GRAS framework (of course)

Evaluation methodology

Apply all evaluations on all reconstructions for several platforms



Measurements

- Bandwidth matrix
- Latency matrix

Algorithms

- Clique
- BW/Lat Spanning Tree
- Improved BW/Lat Tree
- Aggregate

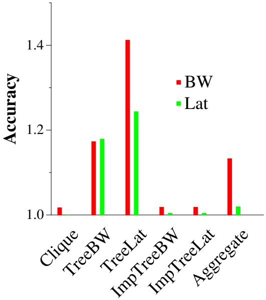
Evaluation criteria

- End-to-end meas.
- Interference count
- Application-level

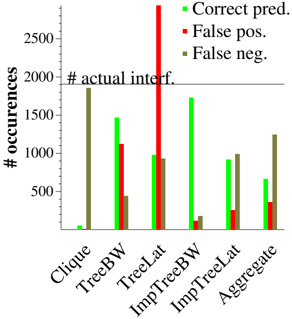
Experiments on simulator: Renater platform

- *Real* platform built manually (real measurements + admin feedback)

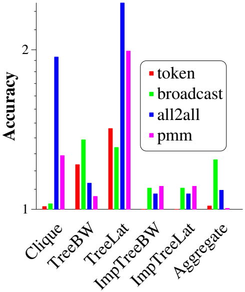
End to end



Interferences



Application-level

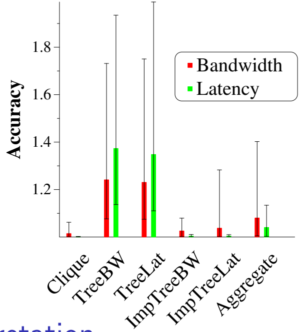


- Clique:
  - Very good for end-to-end (of course)
  - No contention captured $\leadsto$ missing interference $\leadsto$ bad predictions
- Spanning Trees: missing links $\leadsto$ bad predictions
  (over-estimates latency, under-estimates bandwidth, false positive interference)
- Improved Spanning Trees have good predictive power
- Aggregate accuracy discutable

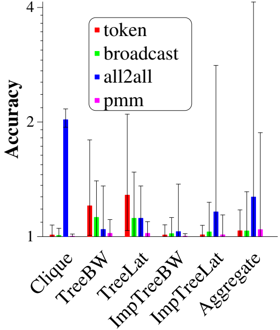
Experiments on simulator: GridG platforms

- GridG is a synthetic platform generator [Lu, Dinda – SuperComputing03]
  Generates *realistic* platforms
- Experiment: 40 platforms (60 hosts – default GridG parameters)

End to end measurements



Application-level measurements



Interpretation

- Naive algorithms lead to poor results
- Improved trees yield good reconstructions
  - ImpTreeBW error ≈ 3% for all2all (worst case)

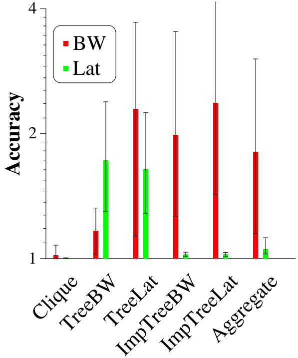
Adding routers to the picture

- New set of experiments: only *leaf* nodes run the measurement processes

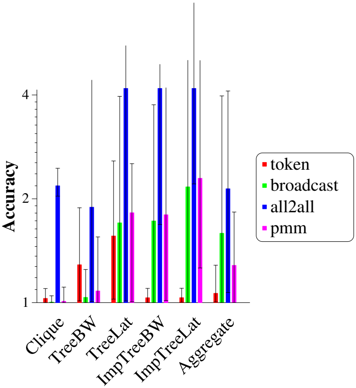
Adding routers to the picture

- New set of experiments: only *leaf* nodes run the measurement processes

End to end measurements



Application-level measurements



Interpretation

- None of the proposed heuristic is satisfactory
- Future work: improve this! Becomes really tricky. Maybe data-minining issue?

Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
  - Associated Tools
    - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
    - Visualization
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

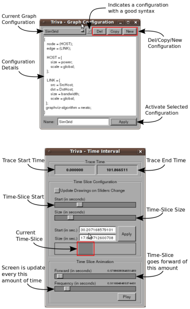
Visualizing SimGrid Simulations with Trivia

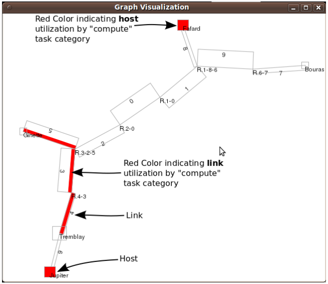
Simulations can produce a *lot* of logs

- Everyone produces ad-hoc parsing scripts
- Not always easy, graphically visualizing more appealing

Building the right visualization tool

- Easy to build a *demoware*: fancy but not really useful
- Trivia: separate (established) project; SimGrid only produces adapted traces





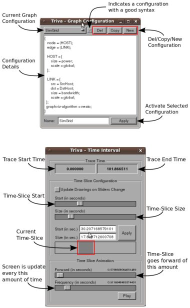
Visualizing SimGrid Simulations with Trivia

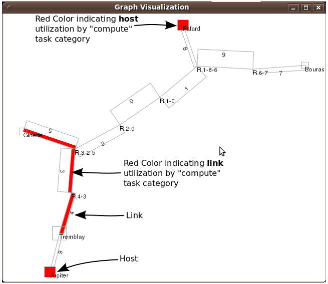
Simulations can produce a *lot* of logs

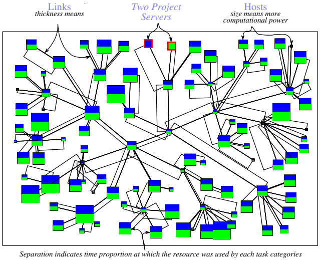
- Everyone produces ad-hoc parsing scripts
- Not always easy, graphically visualizing more appealing

Building the right visualization tool

- Easy to build a *demoware*: fancy but not really useful
- Trivia: separate (established) project; SimGrid only produces adapted traces
- Events, Tasks can be given a application-level semantic category







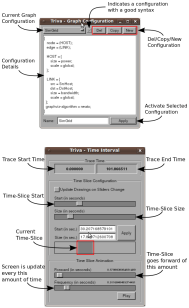
Visualizing SimGrid Simulations with Trivia

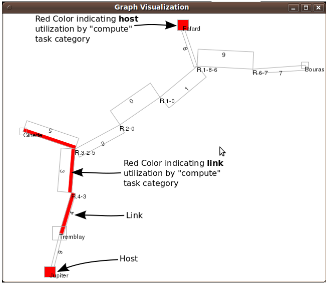
Simulations can produce a *lot* of logs

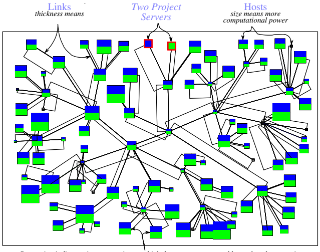
- Everyone produces ad-hoc parsing scripts
- Not always easy, graphically visualizing more appealing

Building the right visualization tool

- Easy to build a *demoware*: fancy but not really useful
- Trivia: separate (established) project; SimGrid only produces adapted traces
- Events, Tasks can be given a application-level semantic category
- Still ongoing effort (integrated in stable releases since spring only)







Separation indicates time proportion at which the resource was used by each task categories

Outline

Conclusions on Distributed Systems Research

Research on Large-Scale Distributed Systems

- Reflexion about common methodologies needed (reproductible results needed)
- Purely theoritical works limited (simplistic settings $\leadsto$ NP-complete problems)
- Real-world experiments time and labor consuming; limited representativity
- Simulation appealing, if results remain validated

Simulating Large-Scale Distributed Systems

- Packet-level simulators too slow for large scale studies
- Large amount of ad-hoc simulators, but discutable validity
- Coarse-grain modelization of TCP flows possible (cf. networking community)
- Model instantiation (platform mapping or generation) remains challenging

SimGrid provides interesting models

- Implements non-trivial coarse-grain models for resources and sharing
- Validity results encouraging with regard to packet-level simulators
- Several orders of magnitude faster than packet-level simulators
- Several models availables, ability to plug new ones or use packet-level sim.

SimGrid provides several user interfaces

SimDag: Comparing Scheduling Heuristics for DAGs of (parallel) tasks

- Declare tasks, their precedences, schedule them on resource, get the makespan

MSG: Comparing Heuristics for Concurrent Sequential Processes

- Declare independent agents running a given function on an host
- Let them exchange and execute tasks
- Easy interface, rapid prototyping; Java, Lua, Ruby bindings
- Also trace-driven simulations (user-defined events and callbacks)

GRAS: Developing and Debugging Real Applications

- *Develop once, run in simulation or in situ* (debug; test on non-existing platforms)
- Resulting application twice slower than MPICH, faster than omniorb
- Highly portable and easy to deploy

SMPI: Running MPI applications on top of SimGrid (beta quality)

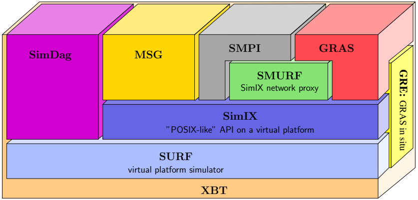
- Runs unmodified MPI code after recompilation (still partial implementation)

Other interfaces possible: OpenMP, BSP-like (any volunteer?)

SimGrid is an active and exciting project

Future Plans

- Better usability: build around simulator (statistics tools, campain management)
- Extreme Scalability for P2P
- Model-checking and semantic debugging
- Emulation solution *à la* MicroGrid



Large community

http://gforge.inria.fr/projects/simgrid/

- 100 subscribers to the user mailling list (40 to -devel)
- 70 scientific publications using the tool for their experiments
- LGPL, 120,000 lines of code (half for examples and regression tests)
- Examples, documentation and tutorials on the web page

Use it in your works!

Grid Simulation and Open Science

Requirement on Experimental Methodology (what do we want)

- Standard methodologies and tools: Grad students learn them to be operational
- Incremental knowledge: Read a paper, Reproduce its results, Improve.
- Reproducible results: Compare easily experimental scenarios
  Reviewers can reproduce result, Peers can work incrementally (even after long time)

Current practices in the field (what do we have)

- Very little common methodologies and tools; *many* home-brewed tools
- Experimental settings rarely detailed enough in literature

These issues are tackled by the SimGrid community

- Released, open-source, stable simulation framework
- Extensive optimization and validation work
- Separation of simulated application and experimental conditions
- Are we there yet? Not quite

SimGrid and Open Science

Simulations are reproducible ... provided that authors ensure that

- Need to publish source code, platform file, statistic extraction scripts . . .
- Almost no one does it. I don't (shame, shame). Why?

SimGrid and Open Science

Simulations are reproducible ... provided that authors ensure that

- Need to publish source code, platform file, statistic extraction scripts ...
- Almost no one does it. I don't (shame, shame). Why?

Technical issues to tackle

- Archiving facilities, Versionning, Branch support, Dependencies management
- Workflows automating execution of test campaigns (`myexperiment.org`)
- We already have most of them (Makefiles, Maven, debs, forges, repositories, ...)
- But still, we don't use it. Is the issue really technical?

SimGrid and Open Science

Simulations are reproducible … provided that authors ensure that

- Need to publish source code, platform file, statistic extraction scripts . . .
- Almost no one does it. I don't (shame, shame). Why?

Technical issues to tackle

- Archiving facilities, Versionning, Branch support, Dependencies management
- Workflows automating execution of test campaigns (`myexperiment.org`)
- We already have most of them (Makefiles, Maven, debs, forges, repositories, . . .)
- But still, we don't use it. Is the issue really technical?

Sociological issues to tackle

- A while ago, simulators were simple, only filling gant charts automatically
- We don't have the culture of reproducibility:
  - "My scientific contribution is the algorithm, not the crappy demo code"
  - But your contribution cannot be assessed if it cannot be reproduced!
- I don't have any definitive answer about how to solve it

Building Open Science Around the Simulator

Going further toward Open Science

- Issues we face in simulation are common to every experimental methodologies
  Test planning, Test Campaign Management, Statistic Extraction
- Tool we need to help Open Science arise in simulation would help others
- Why not step back and try to unit efforts?

What would a perfect world look like?

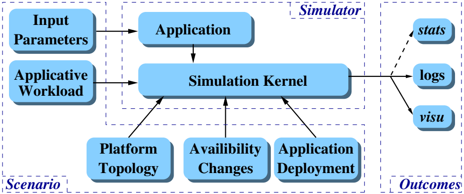
Building Open Science Around the Simulator

Going further toward Open Science

- Issues we face in simulation are common to every experimental methodologies
  Test planning, Test Campaign Management, Statistic Extraction
- Tool we need to help Open Science arise in simulation would help others
- Why not step back and try to unit efforts?

What would a perfect world look like?

A single simulation using SimGrid



An Experiment Campaign on Grid'5000

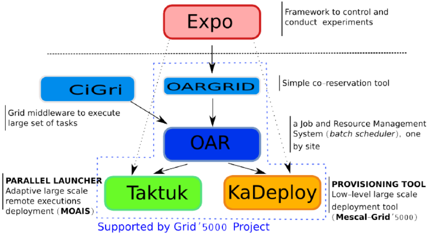


Figure from Olivier Richard

Factorizing is really appealing, even if huge amount of work remains to do

Take-home Messages

HPC and Grid applications tuning and assessment

- Challenging to do; Methodological issues often neglected
- Several methodological ways: in vivo, in vitro, in silico; none perfect

The SimGrid Simulation Framework

- Mature Framework: validated models, software quality assurance
- You should use it!

We only scratched the corner of the problem

- A single simulation is just a brick of the scientific workflow
  - We need more associated tools for campaign management, etc.
- Open Science is a must! (please don't say the truth to physicians or biologists)
  - Technical issues faced, but even more sociological ones
  - Solve it not only for simulation, but for all methodologies at the same time

We still have a large amount in front of us ☺

Question slides

- Implementation of CSPs on top of simulation kernel
- Model-checking GRAS application
- The SimTerpose Project
- Trace Replay: Separate your applicative workload
- XBT from 10,000 feets
- Finding SimGrid's documentation

Implementation of CSPs on top of simulation kernel

Idea

- Each process is implemented in a thread
- Blocking actions (execution and communication) reported into kernel
- A *maestro* thread unlocks the runnable threads (when action done)

Implementation of CSPs on top of simulation kernel

Idea

- Each process is implemented in a thread
- Blocking actions (execution and communication) reported into kernel
- A *maestro* thread unlocks the runnable threads (when action done)

Example

- Thread A:
  - Send "toto" to B
  - Receive something from B
- Thread B:
  - Receive something from A
  - Send "blah" to A

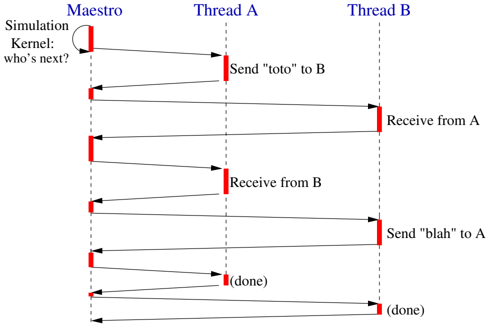
Implementation of CSPs on top of simulation kernel

Idea

- Each process is implemented in a thread
- Blocking actions (execution and communication) reported into kernel
- A *maestro* thread unlocks the runnable threads (when action done)

Example

- Thread A:
  - Send "toto" to B
  - Receive something from B
- Thread B:
  - Receive something from A
  - Send "blah" to A



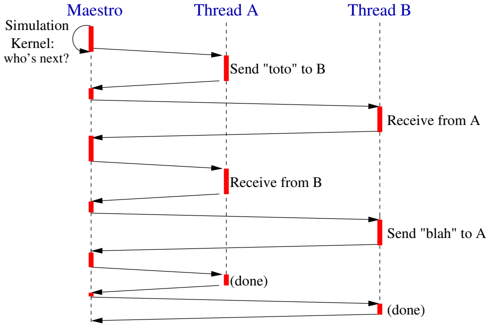
Implementation of CSPs on top of simulation kernel

Idea

- Each process is implemented in a thread
- Blocking actions (execution and communication) reported into kernel
- A *maestro* thread unlocks the runnable threads (when action done)

Example

- Thread A:
  - Send "toto" to B
  - Receive something from B
- Thread B:
  - Receive something from A
  - Send "blah" to A
- Maestro schedules threads
  Order given by simulation kernel
- Mutually exclusive execution
  (don't fear)



Model-checking GRAS application (ongoing work)

Executive Summary

Motivation

- GRAS allows to debug an application on simulator and deploy it when it works
- Problem: when to decide that *it works?*
  - Demonstrate a theorem → conversion to C difficult
  - Test some cases → may still fail on other cases

Model-checking

- Given an initial situation ("we have three nodes"), test all possible executions ("A gets first message first", "B does", "C does", ...)
- Combinatorial search in the tree of possibilities
- Fight combinatorial explosion: cycle detection, symmetry, abstraction

Model-checking in GRAS

- First difficulty: Checkpoint simulated processes (to rewind simulation)
  Induced difficulty: Devise **when** to checkpoint processes
- Second difficulty: Fight against combinatorial explosion

Adding Model-Checking to SimGrid

Difficulties in Distributed System

- Race condition, Deadlock and Starvation, just as in concurrent algorithms
- Lack of global state: only local information available
- Asynchronism: no bound on communication time $\leadsto$ hard to detect failures

$\Rightarrow$ Model-checker for distributed algorithms appealing

Adding Model-Checking to SimGrid

Difficulties in Distributed System

- Race condition, Deadlock and Starvation, just as in concurrent algorithms
- Lack of global state: only local information available
- Asynchronism: no bound on communication time $\rightsquigarrow$ hard to detect failures

$\Rightarrow$ Model-checker for distributed algorithms appealing

But wait a minute...

Wasn't the simulator meant to test distributed algorithm already?!

Do not merge 2 tools in 1 and KISS instead!

Adding Model-Checking to SimGrid

Difficulties in Distributed System

- Race condition, Deadlock and Starvation, just as in concurrent algorithms
- Lack of global state: only local information available
- Asynchronism: no bound on communication time $\rightsquigarrow$ hard to detect failures

$\Rightarrow$ Model-checker for distributed algorithms appealing

But wait a minute...

Wasn't the simulator meant to test distributed algorithm already?!

- Simulation is better than real deployment because it is deterministic
- But possibly very low code coverage
- Model-Checking improves this, and provides counter-examples

$\rightsquigarrow$ Simulation to assess performance, Model-checking to assess correctness

Do not merge 2 tools in 1 and KISS instead!

Adding Model-Checking to SimGrid

Difficulties in Distributed System

- Race condition, Deadlock and Starvation, just as in concurrent algorithms
- Lack of global state: only local information available
- Asynchronism: no bound on communication time $\rightsquigarrow$ hard to detect failures

$\Rightarrow$ Model-checker for distributed algorithms appealing

But wait a minute...

Wasn't the simulator meant to test distributed algorithm already?!

- Simulation is better than real deployment because it is deterministic
- But possibly very low code coverage
- Model-Checking improves this, and provides counter-examples

$\rightsquigarrow$ Simulation to assess performance, Model-checking to assess correctness

Do not merge 2 tools in 1 and KISS instead!

- Avoid manual translation between formalisms to avoid introduction of errors
- Simulator and model-checker both need to:
  - Simulate of the environment (processes, network, messages)
  - Control over the scheduling of the processes
  - Intercept the communication

SimGrid use limitation

Main limitation of SimGrid today

- You have to write your application using its interfaces
- Impossible to reuse it on real life

Some partial solution exist

- GRAS allows you to reuse the code written in SG on real life
  ☹ you still have to learn a new API
- SMPI allows you to run MPI code in SG
  ☹ not anyone use MPI

It ought to be a better solution

- How could I just launch my application on "virtual platform"?

SimGrid use limitation

Main limitation of SimGrid today

- You have to write your application using its interfaces
- Impossible to reuse it on real life

Some partial solution exist

- GRAS allows you to reuse the code written in SG on real life
  ☹ you still have to learn a new API
- SMPI allows you to run MPI code in SG
  ☹ not anyone use MPI

It ought to be a better solution

- How could I just launch my application on "virtual platform"?
- This project (just starting) is dubbed simterpose (interposing a simulator)

Simterpose: presentation (ongoing work)

Goal

- Allowing to use SimGrid on unmodified distributed applications

Why? Motivations

- Test your code in reproducible way
- Dimension your hardware to fit your application
- Benefit of SimGrid associated tools (model-checking? visualization?)
- Process folding (debug in the train – w/o GSM)

How? what's needed?

- (add some sort of launcher – cf. mpirun)
- Intercept any interaction with the system
  send, receive, gettimeofday
- Report them into the simulator
- Reflect simulated reality into real one
  slow down the process by the given amount of time, return simulated clock value

Simterpose: approaches (Ongoing work)

Hard part: interception. How to intercept calls to system and libraries?

- `#define send(a,b,c) sg_send(a,b,c)`
  - ☺ quite easy to do (SMPI does so)
  - ☹ recompilation is mandatory (thus, need source code)
  - ☹ C only
- PTRACE (trace processes as gdb does)
  - ☺ Seamless
  - ☹ syscalls only (one may want to follow pthread calls)
  - ☹ reputed slow
- Library injection (LD_PRELOAD under linux)
  - The system linker use your symbols in preference to classical ones
  - ☹ Only library calls, not syscalls (but anyone uses libcs' wrappers)
  - ☺ Seamless, it could even trick the JVM?
- Valgrind
  - Code injection in binary before running it
  - ☺ Seamless, would trick the JVM
  - ☹ Slow (x40 for empty valgrind tools)
- Real virtual machine (qemu, xen, etc)
  - ☺ Seamless, would trick the JVM
  - ☹ Slow, huge memory requirements for process folding

Trace Replay: Separate your applicative workload

C code

```
static void action_blah(xbt_dynar_t parameters) { ... }
static void action_blih(xbt_dynar_t parameters) { ... }
static void action_bluh(xbt_dynar_t parameters) { ... }
int main(int argc, char *argv[]) {
    MSG_global_init(&argc, argv);
    MSG_create_environment(argv[1]);
    MSG_launch_application(argv[2]);
    /* No need to register functions as usual: actions started anyway */
    MSG_action_register("blah", blah);
    MSG_action_register("blih", blih);
    MSG_action_register("bluh", bluh);

    MSG_action_trace_run(argv[3]); // The trace file to run
}
```

Deployment

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <process host="Tremblay" function="toto"/>
  <process host="Jupiter" function="tutu"/>
  <process host="Fafard" function="tata"/>
</platform>
```

Trace file

```
tutu blah toto 1e10
toto blih tutu
tutu bluh 12
toto blah 12
```

Trace Replay (2/2)

Separating the trace of each process

- Because it's sometimes more convinient (for MPI, you'd have to merge them)
- Simply pass NULL to MSG_action_trace_run()
- Pass the trace file to use as argument to each process in deployment

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <process host="Tremblay" function="toto">
    <argument value="actions_toto.txt"/>
  </process>
  <process host="Jupiter" function="tutu">
    <argument value="actions_tutu.txt"/>
  </process>
</platform>
```

Action Semantic

- This mecanism is completely agnostic: attach the meaning you want to events
- In examples/actions/action.c, we have pre-written event functions for:
  - Basics: send, recv, sleep, compute
  - MPI-specific: isend, irecv, wait, barrier, reduce, bcast, allReduce

XBT from 10,000 feets

C is a basic language: we reinvented the wheel for you

Logging support: Log4C

```
XBT_LOG_NEW_DEFAULT_CATEGORY(test,
    "my own little channel");
XBT_LOG_NEW_SUBCATEGORY(details, test,
    "Another channel");

INFO1("Value: %d", variable);
CDEBUG3(details,"blah %d %f %d", x,y,z);
```

Exception support

```
xbt_ex_t e;
TRY {
  block
} CATCH(e) {
  block /* DO NOT RETURN FROM THERE */
}
```

Debugging your code

- Ctrl-C once: see processes' status
- Press it twice (in 5s): kill simulator

xbt_backtrace_display_current()

```
Backtrace (displayed in thread 0x90961c0):
--->  In master() at masterslave_mailbox.c:35
--->  In ?? ([0x4a69ba5])
```

Advanced data structures

- Hash tables, Dynamic arrays
- FIFOs, Sets, Graphs
- SWAG (but don't use)

String functions

- bprintf: malloc()ing sprintf
- trim, split, subst, diff
- string buffers

Threading support

- Portable wrappers (Lin, Win, Mac, Sim)
- Synchro (mutex, conds, semaphores)

Other

- Mallocators
- Configuration support
- Unit testing (check src/testall)
- Integration tests (tesh: testing shell)

Finding SimGrid's documentation

Finding SimGrid's documentation

User manuals are for wimps

- Real Men read some slides 'cause they are more concise
- They read the examples, pick one modify it to fit their needs
- They may read 2 or 5% of the reference guide to check the syntax
- In doubt, they just check the source code

Finding SimGrid's documentation

User manuals are for wimps

- Real Men read some slides 'cause they are more concise
- They read the examples, pick one modify it to fit their needs
- They may read 2 or 5% of the reference guide to check the syntax
- In doubt, they just check the source code

lusers don't read the manual either

- Proof: that's why the RTFM expression were coined out
- Instead, they always ask same questions to lists, and get pointed to the FAQ

Finding SimGrid's documentation

User manuals are for wimps

- Real Men read some slides 'cause they are more concise
- They read the examples, pick one modify it to fit their needs
- They may read 2 or 5% of the reference guide to check the syntax
- In doubt, they just check the source code

lusers don't read the manual either

- Proof: that's why the RTFM expression were coined out
- Instead, they always ask same questions to lists, and get pointed to the FAQ

So, where is all SimGrid documentation?

- The SimGrid tutorial is a 200 slides presentation (motivation, models, example of use, internals)
- Almost all features of UAPI are demoed in an example (coverage testing)
- The reference guide contains a lot in introduction sections (about XBT)
- The FAQ contains a lot too (installing, visu, XML, exotic features)
- The code is LGPL anyway