Simulation for Experimenting HPC Systems

Martin Quinson (Nancy University, France)
et Al.

Nancy, June 3 2010

Scientific Computation Applications

Classical Approaches in science and engineering

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

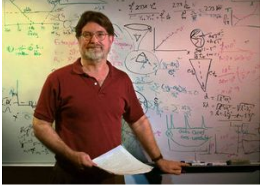
That's not always desirable (or even possible)

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

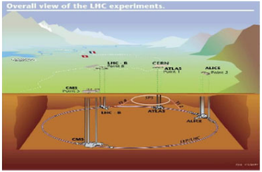
The third scientific way: *Computational Science*

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

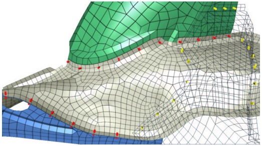
⤳ High Performance Computing Systems



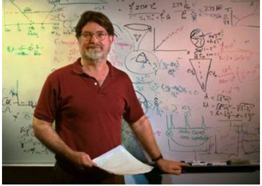
Georges Smoot
Physics Nobel Price 1996



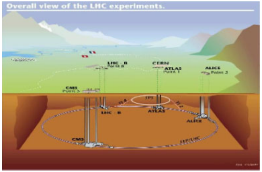
Large Hardron Collider



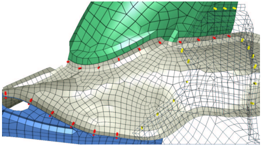
Scientific Computation Applications



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

⇝ 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*

Studying Large Distributed HPC Systems (Grids)

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 ($\approx$ in vivo): Real applications on Real platforms
  ☺ Believable; ☹ Hard and long. Experimental control? Reproducibility?
- Emulation ($\approx$ 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

$\Rightarrow$ No approach is enough, all are mandatory

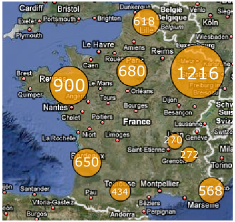
Outline

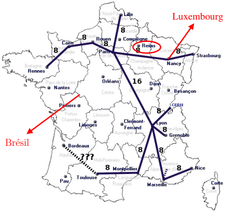
In vivo approach to HPC experiments (direct experiment)

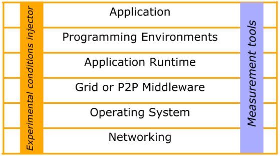
- Principle: Real applications, controlled environment
- Challenges: Hard and long. Experimental control? Reproducibility?

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

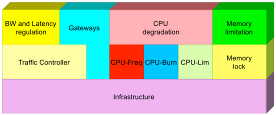
- PlanetLab: No experimental control $\Rightarrow$ no reproducibility
- Production Platforms (EGEE): must use provided middleware
- FutureGrid: future American experimental platform inspired from Grid'5000

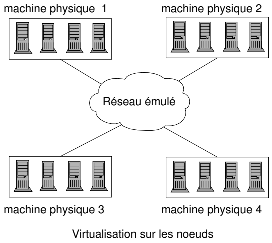
In vitro approach to HPC experiments (emulation)

- Principle: Injecting load on real systems for the experimental control
  $\approx$ *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





Other existing tools

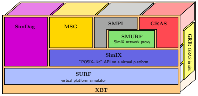
- Network emulation: ModelNet, DummyNet, . . .
  Tools rather mature, but limited to network
- Applicative emulation: MicroGrid, eWan, Emulab
  Rarely (never?) used outside the lab where they were created

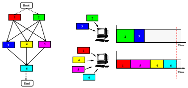
In silico approach to HPC experiments (simulation)

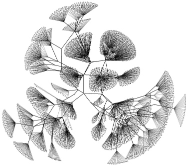
- Principle: Prototypes of applications, models of platforms
- Challenges: Get realistic results (experimental bias)

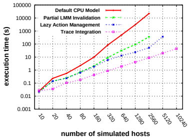
SimGrid: generic simulation framework for distributed applications

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







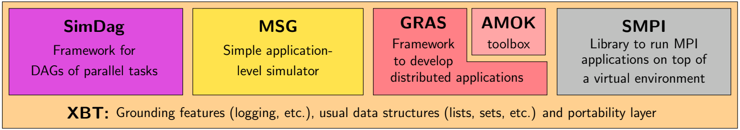


Other existing tools

- *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

Outline

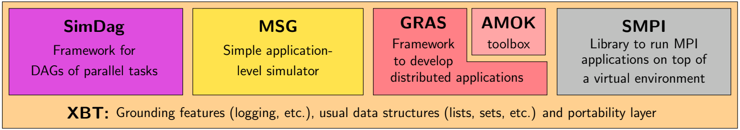
User-visible SimGrid Components



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

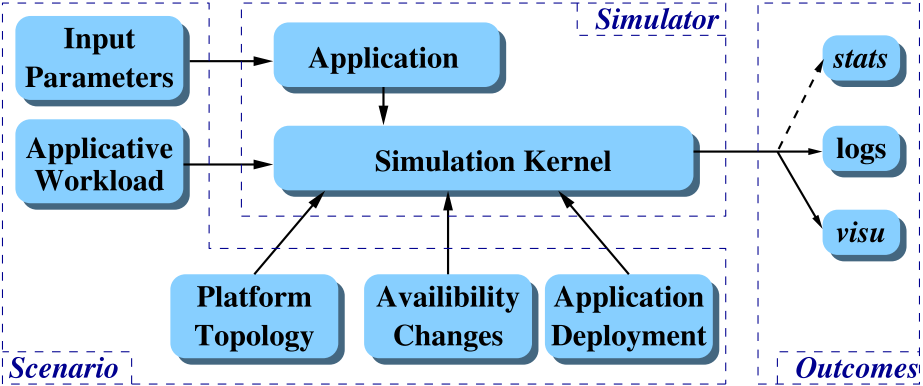
Executive summary, but representative

- 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

Under the Hood: Simulation Models

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: $T = \lambda + \frac{size}{\beta}$; Better: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

Under the Hood: Simulation Models

Modeling CPU

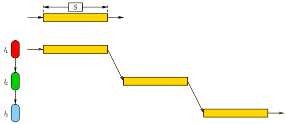
- Resource delivers *pow* flop / sec; task require *size* flop ⇒ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

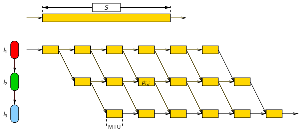
Modeling Single-Hop Networks

- Simplistic: $T = \lambda + \frac{size}{\beta}$; Better: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

Modeling Multi-Hop Networks

- Simplistic Models: Store & Forward or Wormhole





☺ Easy to implement; ☹ Not realistic *(TCP Congestion omitted)*

Under the Hood: Simulation Models

Modeling CPU

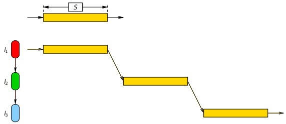
- Resource delivers *pow* flop / sec; task require *size* flop $\Rightarrow$ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

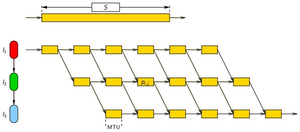
Modeling Single-Hop Networks

- Simplistic: $T = \lambda + \frac{size}{\beta}$; Better: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ *(TCP windowing)*

Modeling Multi-Hop Networks

- Simplistic Models: Store & Forward or Wormhole





☺ Easy to implement; ☹ Not realistic *(TCP Congestion omitted)*

- NS2 and other packet-level study the path of each and every network packet

☺ Realism commonly accepted; ☹ Sloooooow

Analytical Network Models

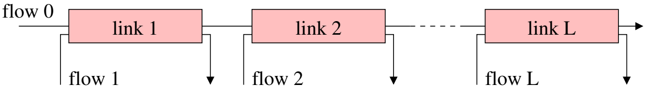
TCP bandwidth sharing studied by several authors

- 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
  $\rightsquigarrow$ 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

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

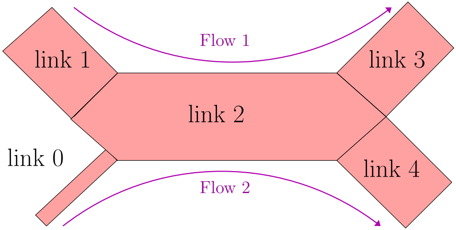
L. Massoulié and J. Roberts, *Bandwidth sharing: objectives and algorithms*, IEEE/ACM Trans. Netw., vol. 10, no. 3, pp. 320-328, 2002.

Max-Min Fairness Computation: Backbone Example

Algorithm: loop on these steps

- search for the bottleneck link (so that share of its flows is minimal)
- set all flows using it
- remove the link

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



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

$\lambda_1 =$

$\lambda_2 =$

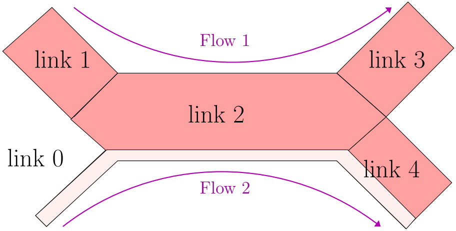
- The limiting link is 0

Max-Min Fairness Computation: Backbone Example

Algorithm: loop on these steps

- search for the bottleneck link (so that share of its flows is minimal)
- set all flows using it
- remove the link

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



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

$\lambda_1 =$
$\lambda_2 = 1$

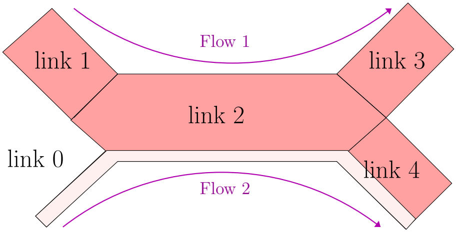
- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links

Max-Min Fairness Computation: Backbone Example

Algorithm: loop on these steps

- search for the bottleneck link (so that share of its flows is minimal)
- set all flows using it
- remove the link

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



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

$\lambda_1 =$
$\lambda_2 = 1$

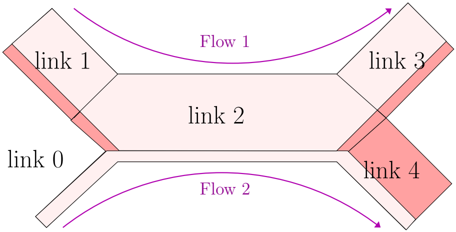
- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links
- The limiting link is 2

Max-Min Fairness Computation: Backbone Example

Algorithm: loop on these steps

- search for the bottleneck link (so that share of its flows is minimal)
- set all flows using it
- remove the link

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



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

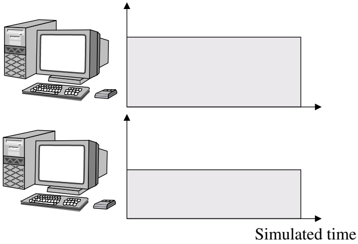
$\lambda_1 = 999$
$\lambda_2 = 1$

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

How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

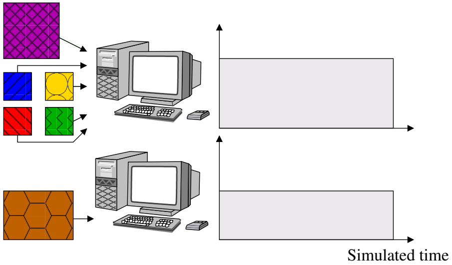


How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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

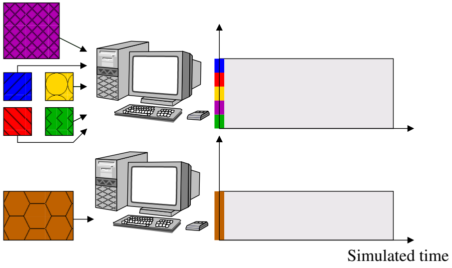


How are these models used in practice?

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)

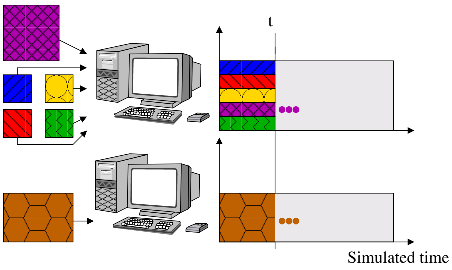


How are these models used in practice?

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

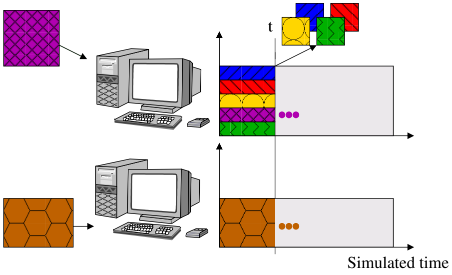


How are these models used in practice?

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

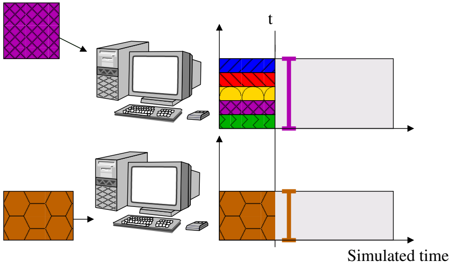


How are these models used in practice?

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

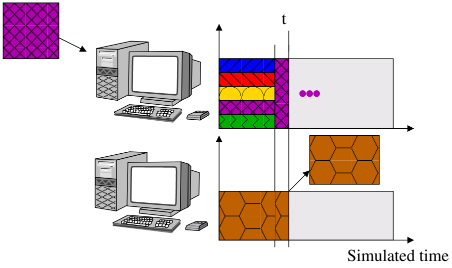


How are these models used in practice?

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

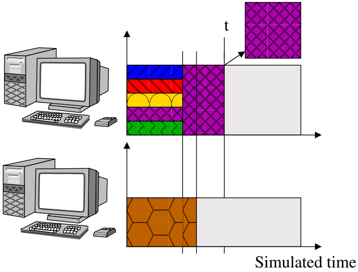


How are these models used in practice?

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

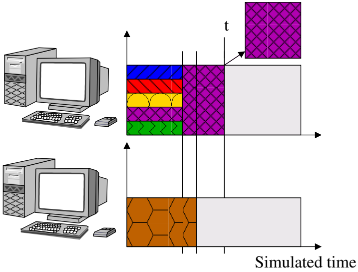


How are these models used in practice?

Simulation kernel main loop

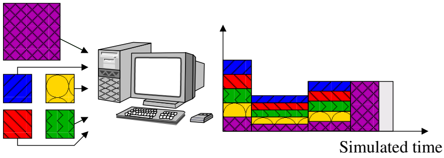
Data: set of resources with working rate

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)

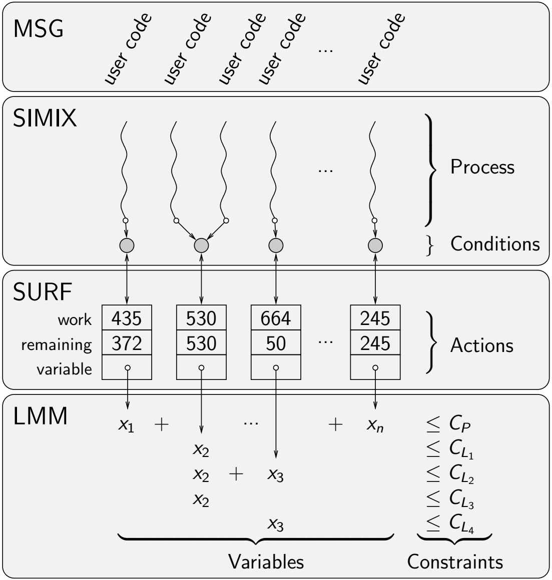
SimGrid Internals in a Nutshell for Users

SimGrid Layers

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

Changing the Model

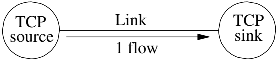
- "--cfg=network_model"
- Several fluid models
- Several constant time
- GTNetS wrapper
- Build your own



Outline

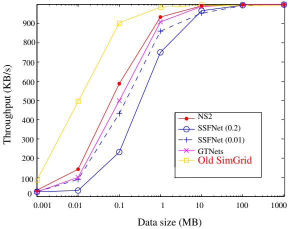
Validation experiments on a single link

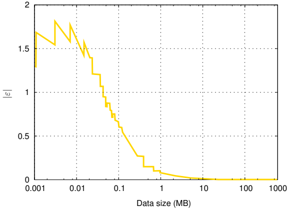
Experimental settings



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

Evaluation Results



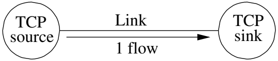


- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default
- GTNetS is equally distant from others

- Old SimGrid model omitted slow start effects

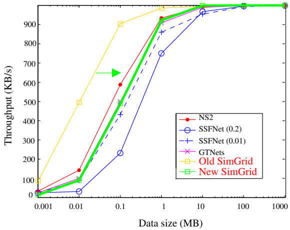
Validation experiments on a single link

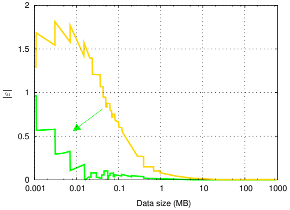
Experimental settings



- 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

$\leadsto$ Better instantiation of MaxMin model

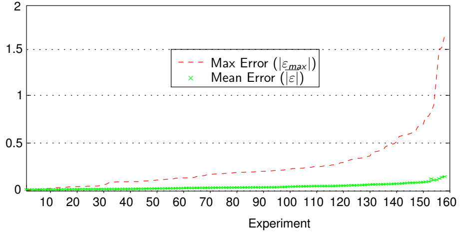
$\beta'' \leadsto .92 \times \beta'; \lambda \leadsto 10.4 \times \lambda$

- Resulting validity range quite acceptable

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

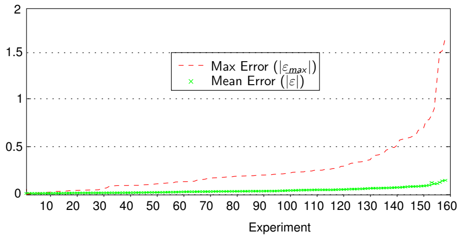
Validation experiments on random platforms

- 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%



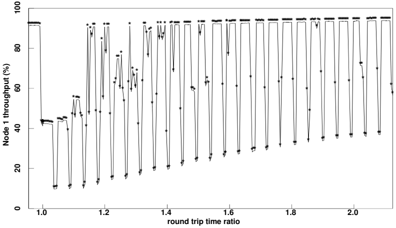
Validation experiments on random platforms

- 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)



$\rightsquigarrow$ We're adding random jitter to continue SimGrid validation

Simulation scalability assessment

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

$\Rightarrow$ at most 10,922 user processes

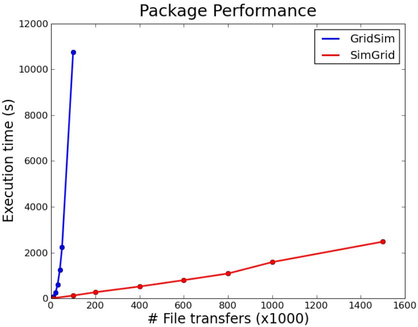
†: out of memory

Simulation scalability assessment

During Summer 2009, 2 interns @CERN evaluated grid simulators

- Attempted to simulate one day on 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



Outline

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

HPC experiments and Open Science

Going further

- Issues we face in simulation are common to every experimental methodologies
- 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?

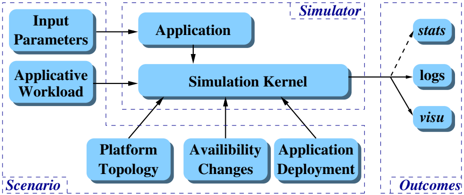
HPC experiments and Open Science

Going further

- Issues we face in simulation are common to every experimental methodologies
- 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 simulation using SimGrid



An experiment on Grid'5000

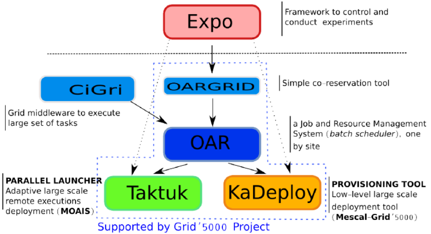


Figure from Olivier Richard

Basic ideas are the same, even if huge amount of work ahead to factorize

Conclusions

HPC and Grid applications tuning and assessment

- Challenging to do; Several methodological ways: in vivo, in vitro, in silico
- No methodology sufficient, all needed together

The SimGrid simulation framework

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

We only scratched the corner of the problem

- 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 ☺