# Versatile yet Scalable and Accurate Simulation of Distributed Applications and Systems: The SimGrid Project

Arnaud Legrand *et Al.*

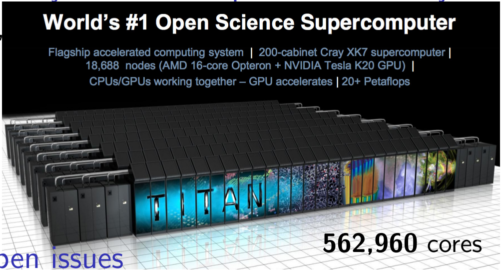
Grenoble University, CNRS, France

SimuTools. Cannes, France
March 07 2013

# Large-Scale Distributed Systems Research

## Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing



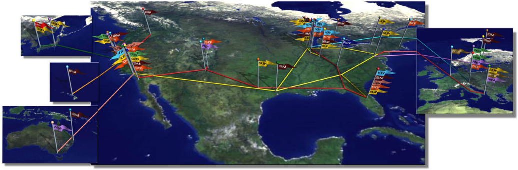

## Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

# Large-Scale Distributed Systems Research

## Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing



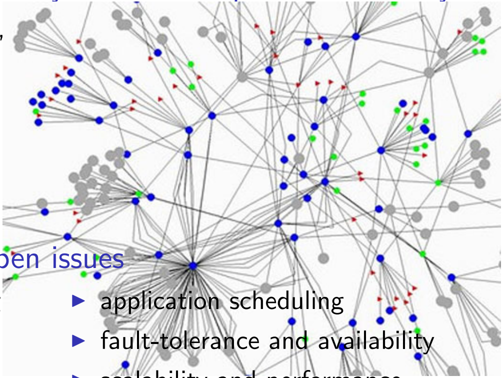
## Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

# Large-Scale Distributed Systems Research

Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing



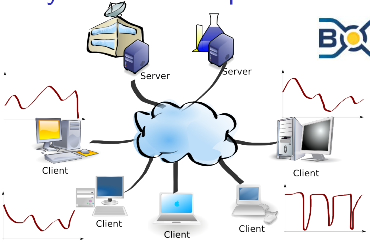
Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

# Large-Scale Distributed Systems Research

## Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing




BOINC

**540,000+** hosts
**7.3+** PetaFLOPS

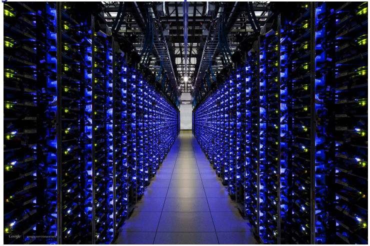
## Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

# Large-Scale Distributed Systems Research

## Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing



Google Data Center

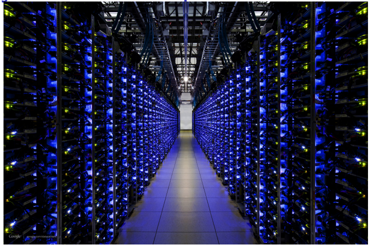
## Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

# Large-Scale Distributed Systems Research

## Large-scale parallel and distributed systems are in production today

- HPC (clusters, petascale systems, soon exascale...)
- Grid platforms
- Peer-to-peer file sharing
- Distributed volunteer computing
- Cloud Computing



Google Data Center

## Complex platforms with many open issues

- resource discovery and monitoring
- resource & data management
- energy consumption reduction
- resource economics
- application scheduling
- fault-tolerance and availability
- scalability and performance
- decentralized algorithms

## Such **applications** and **systems** deserve very advanced analysis

- Their debugging and tuning are technically difficult
- Their use induce high methodological challenges

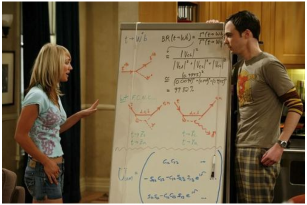
# Methodological Approaches

## 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, fail to capture key characteristics of real systems



The Big Bang Theory

## ⇒ One must run experiments

⇝ Most published research in the area is experimental

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

# 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.
- ☹ Experiments can be too expensive, slow, dangerous

Large Hadron Collider

# In vivo approach to HPC experiments (direct experiment)


Large Hadron Collider

- ☺ Eminently *believable* to demonstrate the proposed approach applicability.
- ☹ Experiments can be too expensive, slow, dangerous
- ☹ Very time and labor consuming
  - Entire application must be functional
- ☹ 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?)
- ☹ Real experiments are often 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)

Large Hadron Collider

- ☺ Eminently *believable* to demonstrate the proposed approach applicability.
- ☹ Experiments can be too expensive, slow, dangerous
- ☹ Very time and labor consuming
  - Entire application must be functional
- ☹ 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?)
- ☹ Real experiments are often 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!**

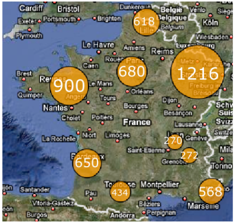
# Example of Tools for Direct Experimentation

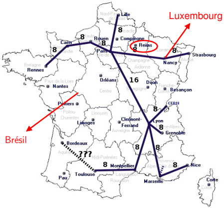
- Principle: Real applications, controlled environment
- Challenges: Hard and long. Experimental control? Reproducibility?

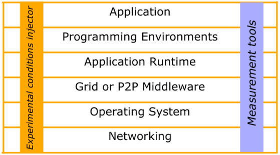
## Grid'5000 project: a scientific instrument for the HPC

- 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

# Emulation (in vitro) as an Experimental Methodology

## Execute your application in a perfectly controlled environment

- Real platforms are not controllable, so how to achieve this?
- Let's look at what engineers do in other fields

# Emulation (in vitro) as an Experimental Methodology

## Execute your application in a perfectly controlled environment

- Real platforms are not controllable, so how to achieve this?
- Let's look at what engineers do in other fields

When you want to build a race car ...adapted to wet tracks in a dry country

# Emulation (in vitro) as an Experimental Methodology

## Execute your application in a perfectly controlled environment

- Real platforms are not controllable, so how to achieve this?
- Let's look at what engineers do in other fields

When you want to build a race car

...adapted to wet tracks

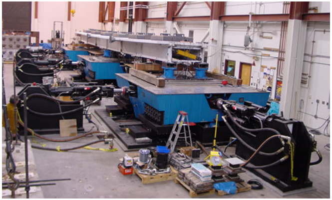
in a dry country

Why don't you just control the climate?

or tweak the car's reality?

# Emulation in other Sciences

Studying earthquake effects on bridges



Studying tsunamis

Studying Coriolis effect and stratification vs. viscosity

Studying climate change effects on ecosystems

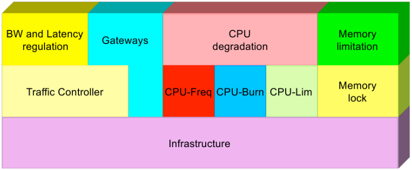
(who said that science is not fun??)

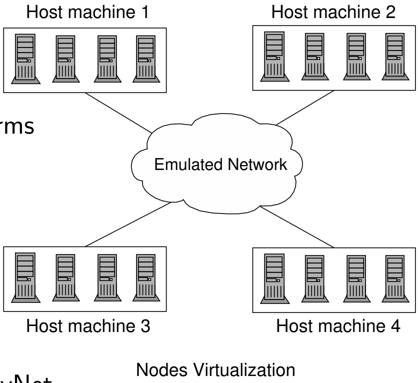
# In vitro approach to HPC experiments (emulation)

- 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, control often induces bias

## Wrekavoc: applicative emulator

- Emulates CPU and network
- Homogeneous or heterogeneous platforms



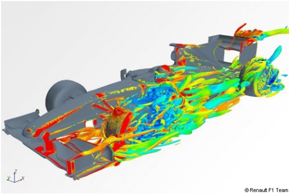

## Other existing tools

- 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)

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

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



Car Mesh

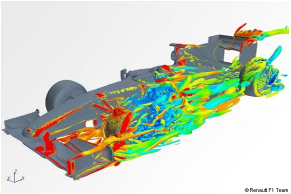
## Simulation in a nutshell

*Computer prediction of the behavior of a system using a (approximate) model*

# 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

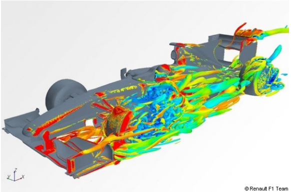
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results




Car Mesh

## Simulation in a nutshell

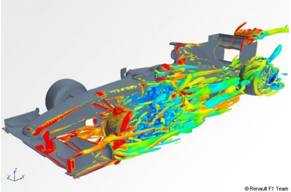
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

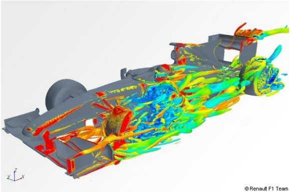
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

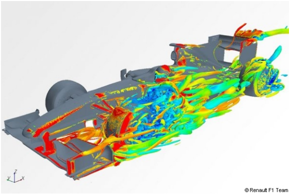
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

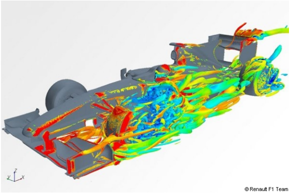
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results




Car Mesh

## Simulation in a nutshell

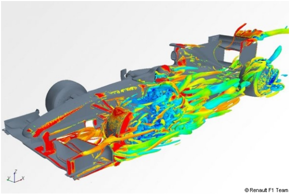
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

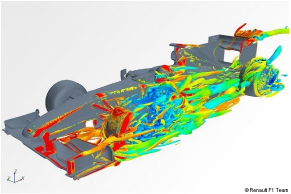
*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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

# 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
- Conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results



Car Mesh

## Simulation in a nutshell

*Computer prediction of the behavior of a system using a (approximate) model*

- Model: Set of equations; Objects whose state evolution is governed by a set of rules; ...
- Simulator: Program solving equations or 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

Versatility

# 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
  - You can read a paper,
  - reproduce a subset of its results,
  - improve

  Workshop on Duplicating, Deconstructing, and Debunking

## 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

# Simulation in Distributed Systems Research

## Little common methodologies and tools

- Experimental settings rarely detailed enough in literature
- No established simulator up until a few years ago
- Simulators are short-lived and rarely made available
- 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 Distributed Systems Research

## Little common methodologies and tools

- Experimental settings rarely detailed enough in literature
- No established simulator up until a few years ago
- Simulators are short-lived and rarely made available
- 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

## Why?

- Understanding and controlling the simulator code is important.
- Researchers lack trust in a simulator developed by others. . .

# Simulation in Distributed Systems Research

## Little common methodologies and tools

- Experimental settings rarely detailed enough in literature
- No established simulator up until a few years ago
- Simulators are short-lived and rarely made available
- 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

## Why?

- Understanding and controlling the simulator code is important.
- Researchers lack trust in a simulator developed by others...
- ...or researchers don't care. All they want is a paper.

## Consequence

Most published simulation results are impossible to reproduce by researchers other than their authors

Yet, simulation results should be easily repeatable by design!

# The Specialization Excuse

## But again... Why ?

- Most simulators are are domain-specific (P2P, HPC, grid, cloud, . . . ).

  One simulator to rule them all?
- Although many simulators claim to be generic, they were developed with a specific purpose in mind and can hardly be used beyond their initial purpose.
- Hence, simulators are developed by researchers for their own research field and these researchers are domain experts, not simulation experts.

# The Specialization Excuse

## But again... Why ?

- Most simulators are are domain-specific (P2P, HPC, grid, cloud, ...).

  One simulator to rule them all?
- Although many simulators claim to be generic, they were developed with a specific purpose in mind and can hardly be used beyond their initial purpose.
- Hence, simulators are developed by researchers for their own research field and these researchers are domain experts, not simulation experts.

## Popular Wisdom 1

*Simulators are toys that any MSc. C.S. student can write.* ☺

## Popular Wisdom 2

**Specialization** *allows for "better" simulation, i.e., simulations that achieve a desired trade-off between* **accuracy** *(low simulation error) and* **scalability** *(ability to run big and/or fast simulations).*

# The SimGrid Project

## SimGrid: a generic simulation framework for distributed applications

- 13 years old open-source project. Collaboration between
  - France (INRIA, CNRS, Univ. Lyon, Nancy, Grenoble, ...)
  - USA (UCSD, U. Hawaii), ...
- Started like others (unsatisfied with practice, no simulation specialists):

  Wouldn't it be possible to have both **accuracy**, **scalability** and **versatility**?
- Scalable (time and memory), modular, portable. +140 publications.

## Other existing tools

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

## Purpose of this talk

- Present some efforts and results obtained in the SimGrid project related to improving accuracy, scalability and versatility.
- Explain how it compares to other domain-specific simulators.

# Agenda

- Experiments for Large-Scale Distributed Systems Research
  Main Methodological Approaches: In Vivo, In Silico, In Vitro
  Bad Practices in Large-Scale Distributed Systems Research

- The SimGrid Project
  User Interface(s)
  How accurate? The Validation Quest
  How big and how fast ?

- Conclusions
  Keynote Recap
  Going Further: Experiment planning and Open Science
  Take-home Messages

# Outline

# User-visible SimGrid Components

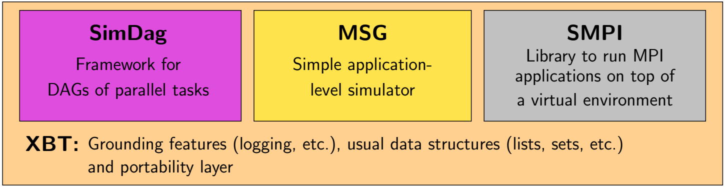
| SimDag | MSG | SMPI |
|---|---|---|
| Framework for DAGs of parallel tasks | Simple application-level simulator | Library to run MPI applications on top of a virtual environment |

**XBT:** Grounding features (logging, etc.), usual data structures (lists, sets, etc.) and portability layer

## SimGrid user APIs

- SimDag: study heuristics handling DAG of (parallel) tasks
- MSG: model applications as Concurrent Sequential Processes (Java/Ruby/Lua bindings available)
- SMPI: simulate MPI codes

# User-visible SimGrid Components




## SimGrid user APIs

- SimDag: study heuristics handling DAG of (parallel) tasks
- MSG: model applications as Concurrent Sequential Processes (Java/Ruby/Lua bindings available)
- SMPI: simulate MPI codes

## Which API should I choose?

- Your study scheduling of DAG structured applications $\leadsto$ SimDag
- You have an MPI code to study $\leadsto$ SMPI
- Your system comprises concurrent processes with possibly complex interactions $\leadsto$ MSG
- 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: location independant communication channel

# 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: location independant communication channel

# 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

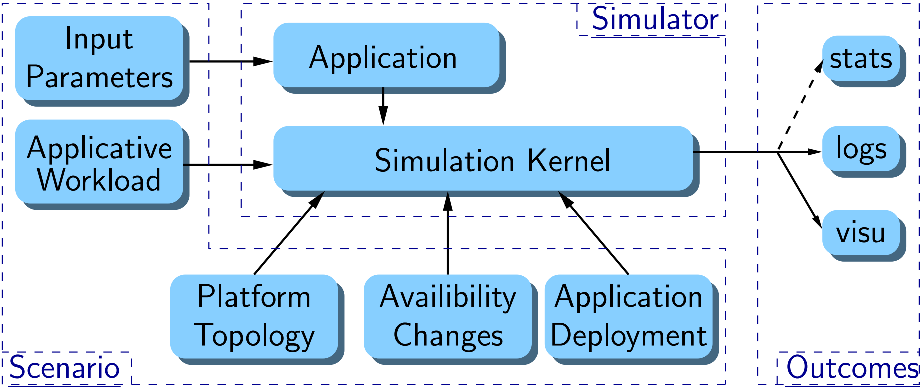
## Executive summary, but representative

- Similar in others interfaces, but:
  - glue is generated by a script in SMPI 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 toolkit**

Such organization favors versatility and decouples application modeling from platform modeling

# Outline

- Experiments for Large-Scale Distributed Systems Research
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Bad Practices in Large-Scale Distributed Systems Research
- The SimGrid Project
  - User Interface(s)
  - How accurate? The Validation Quest
  - How big and how fast ?
- Conclusions
  - Keynote Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

# 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

Along the same lines: Weaver and MsKee, *Are Cycle Accurate Simulations a Waste of Time?*, Proc. of the Workshop on Duplicating, Deconstruction and Debunking, 2008

# Network Communication Models

**Packet-level simulation** Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,. . . )

- full simulation of the whole protocol stack
- complex models $\leadsto$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

# Network Communication Models

**Packet-level simulation** Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,...)

- full simulation of the whole protocol stack
- complex models $\leadsto$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

**Delay-based models** The simplest ones...

- communication time = constant delay, statistical distribution, LogP $\leadsto$($\Theta(1)$ footprint and $O(1)$ computation)
- coordinate based systems to account for geographic proximity $\leadsto$($\Theta(N)$ footprint and $O(1)$ computation)

Although very scalable, these models ignore network congestion and typically assume large bisection bandwidth

# Network Communication Models (cont'd)

## Flow-level models

A communication is simulated as a single entity (like a flow in pipes):

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

# Network Communication Models (cont'd)

## Flow-level models

A communication is simulated as a single entity (like a flow in pipes):

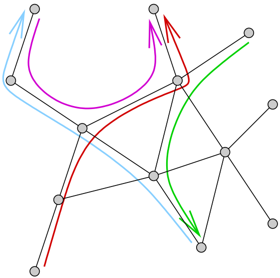
$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

Assume steady-state and **share bandwidth** every time a new flow appears or disappears

Setting a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

Constraints For all link $j$: $\displaystyle\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$



# Network Communication Models (cont'd)

## Flow-level models

A communication is simulated as a single entity (like a flow in pipes):

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

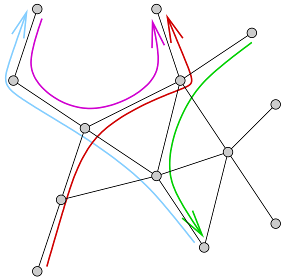
Assume steady-state and **share bandwidth** every time a new flow appears or disappears

**Setting** a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

**Constraints** For all link $j$: $\displaystyle\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$

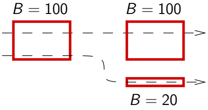
**Objective function**

- Max-Min $\max(\min(\varrho_i))$
- or other fancy objectives
  e.g., Reno $\sim \max(\sum \arctan(\varrho_i))$
  Vegas $\sim \max(\sum \log(\varrho_i))$



# Invalidating Simulators from the Litterature

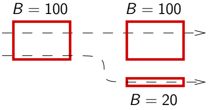
## Naive flow models documented as wrong

| Setting | Expected Output | Output |
|---|---|---|
|  | | |

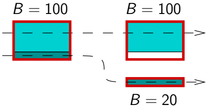
# Invalidating Simulators from the Litterature

## Naive flow models documented as wrong

Setting



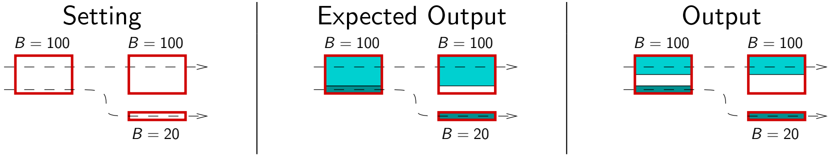

Expected Output




Output

# Invalidating Simulators from the Litterature

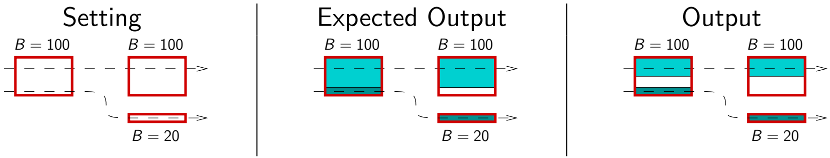
## Naive flow models documented as wrong




Known issue in Narses (2002), OptorSim (2003), GroudSim (2011).

# Invalidating Simulators from the Litterature

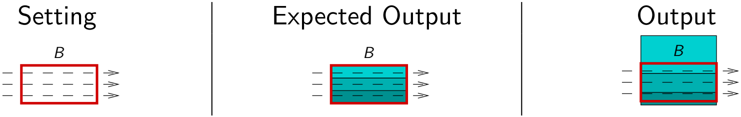
## Naive flow models documented as wrong




Known issue in Narses (2002), OptorSim (2003), GroudSim (2011).

## Validation by general agreement

> *"Since SimJava and GridSim have been extensively utilized in conducting cutting edge research in Grid resource management by several researchers, bugs that may compromise the validity of the simulation have been already detected and fixed."* CloudSim, ICPP'09




Buggy flow model (GridSim 5.2, Nov. 25, 2010). Similar issues with naive packet-level models.

# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.

# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.

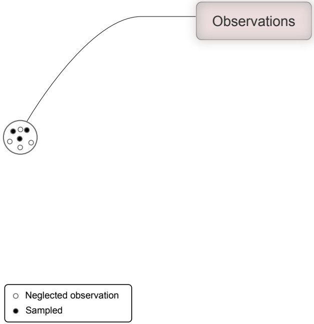
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



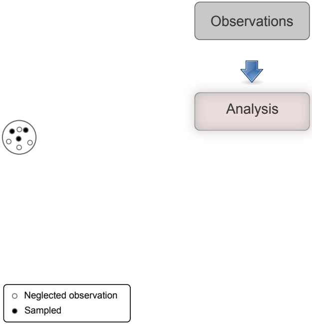
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



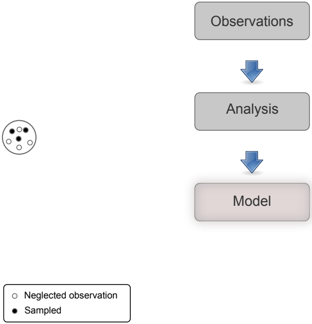
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



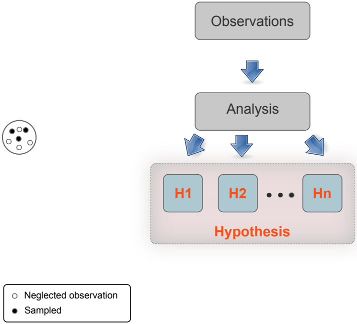
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



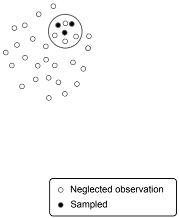
# Validation vs. Invalidation

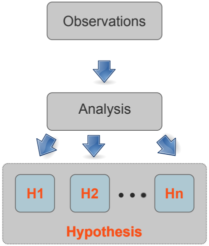
## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



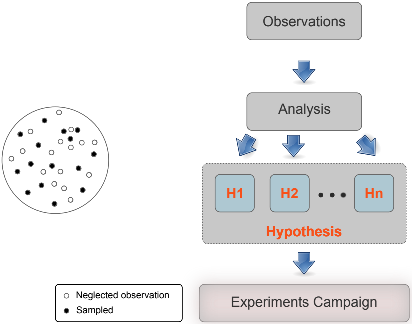
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



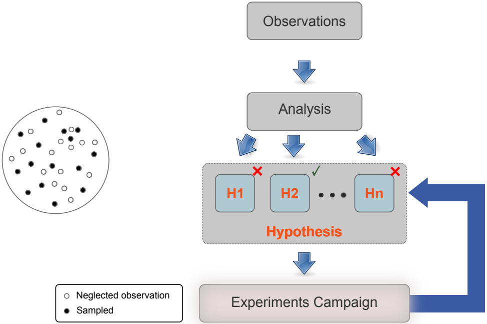
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



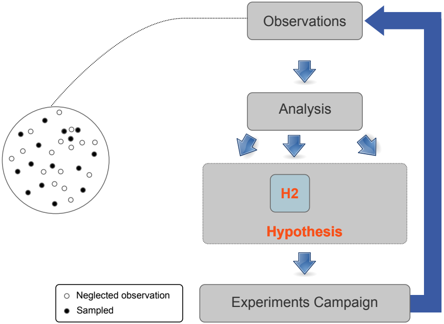
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



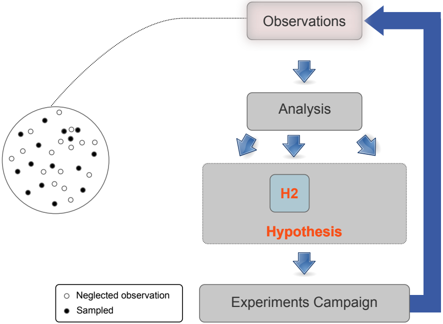
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



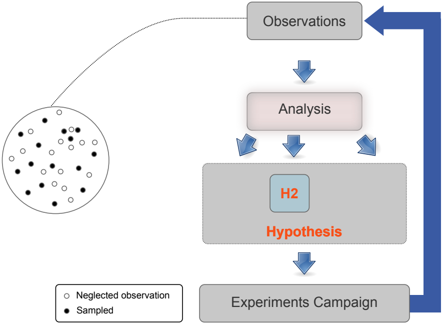
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  ⇝ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



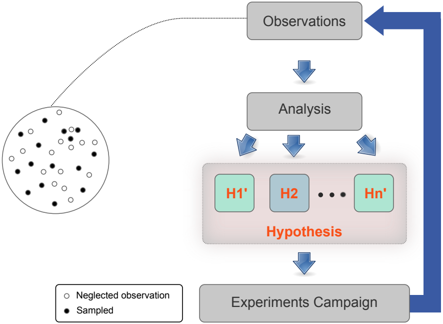
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\rightsquigarrow$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



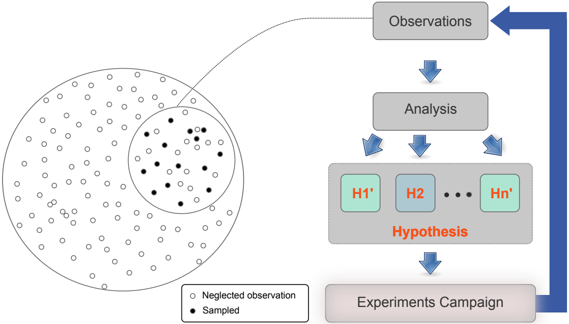
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

Other sciences assess the quality of a model by trying to invalidate it.



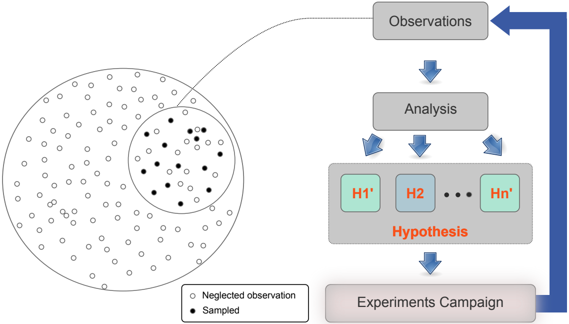
# Validation vs. Invalidation

## Validation

- Articles full of "convincing" graphs but shallow description, unavailable or broken code
- Optimistic validation, i.e., only for a few cases in which the model is expected to work well
  $\leadsto$ merely verifies that the model implementation is correct and that its results are not completely unreasonable

## Invalidation and crucial experiments

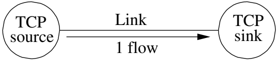
Other sciences assess the quality of a model by trying to invalidate it.




1. A cyclic process
2. Experiments should be designed to objectively prove or disprove an hypothesis
3. Rejected hypothesis provide generally much more insight than accepted ones

# Wanted Feature (1): Flow Control Limitation

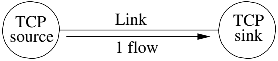
## Experimental settings




- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

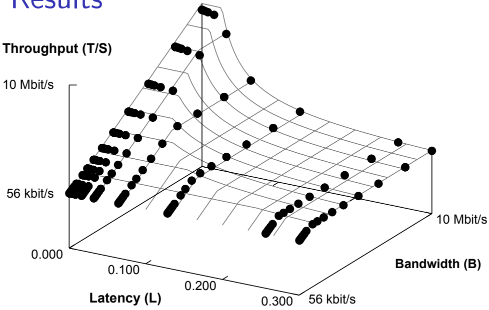
# Wanted Feature (1): Flow Control Limitation

## Experimental settings




- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

## Results




## Legend

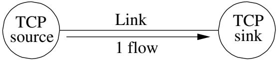
- Mesh: SimGrid results

$$\frac{S}{S/min(B, \frac{W}{2L}) + L}$$

- •: GTNetS results

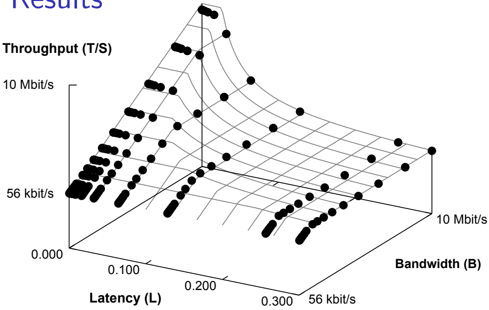
# Wanted Feature (1): Flow Control Limitation

## Experimental settings




- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

## Results




## Legend

- Mesh: SimGrid results

$$\frac{S}{S/min(B, \frac{W}{2L}) + L}$$

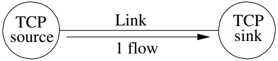
- •: GTNetS results

## Conclusion

- SimGrid estimations close to packet-level simulators (when S=100MB)
  - When $B < \frac{W}{2L}$ (B=100KB/s, L=500ms), $|\varepsilon_{max}| \approx \overline{|\varepsilon|} \approx 1\%$
  - When $B > \frac{W}{2L}$ (B=100KB/s, L= 10ms), $|\varepsilon_{max}| \approx \overline{|\varepsilon|} \approx 2\%$

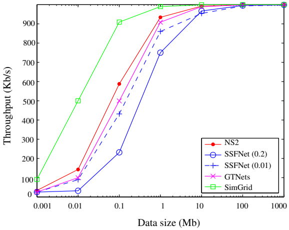
# Wanted Feature (2): Slow Start

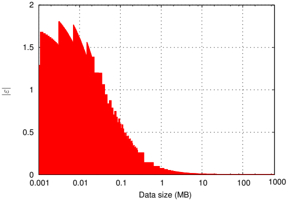
## Experimental settings




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

## Evaluation of the SimGrid fluid model

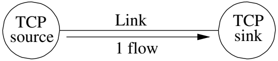



- Packet-level tools don't completely agree

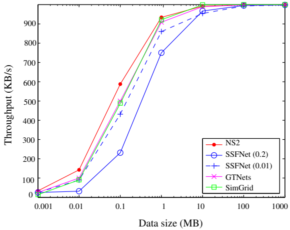
# Wanted Feature (2): Slow Start

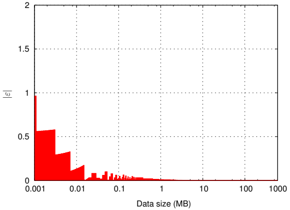
## Experimental settings




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

## Evaluation of the SimGrid fluid model




- Packet-level tools don't completely agree
- Statistical analysis of GTNetS slow-start
- Better instantiation
  - Bandwidth decreased (97%)
  - Latency changed to $13.1 \times L$
  - Hence: $Time = \frac{S}{min\left(0.97 \times B, \frac{W}{2L}\right)} + 13.1 \times L$
- This dramatically improve validity range compared to using raw $L$ and $B$

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

# Wanted Feature (3): RTT-unfairness

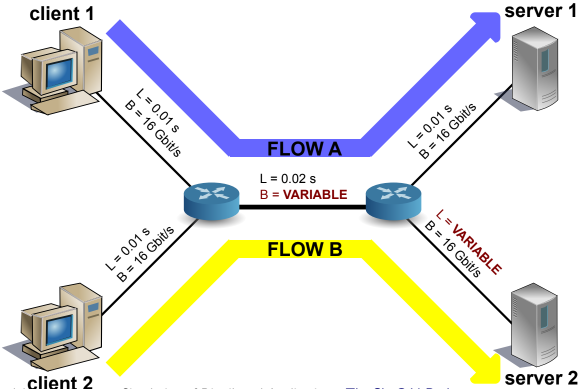
**Hypothesis:** **Bottleneck links are proportionally shared with respect to flow RTT**

# Wanted Feature (3): RTT-unfairness

**Hypothesis:** **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \textbf{ where } RTT_i \approx \sum_{\text{flow i uses link j}} (L_j) \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth

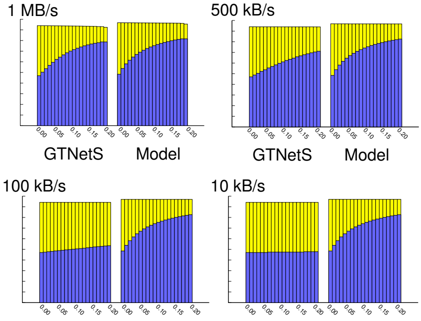


# Wanted Feature (3): RTT-unfairness

**Hypothesis: Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \textbf{ where } RTT_i \approx \sum_{\text{flow i uses link j}} (L_j) \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth

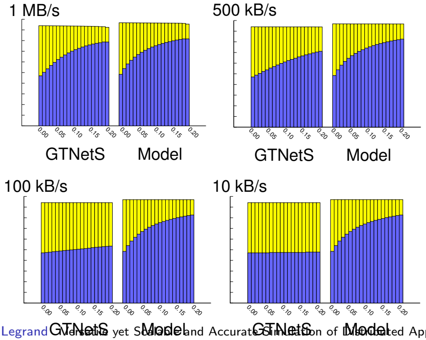


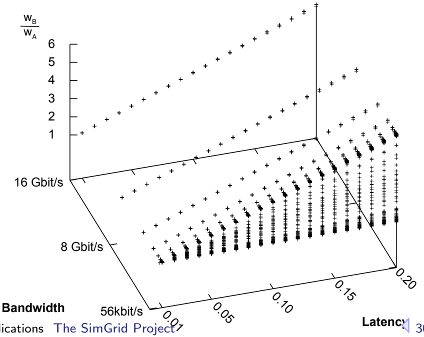
# Wanted Feature (3): RTT-unfairness

**Hypothesis:** **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \text{ where } \cancel{RTT_i \approx \sum_{\text{flow i uses link j}} (L_j)} \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth
- However, bandwidth also matters



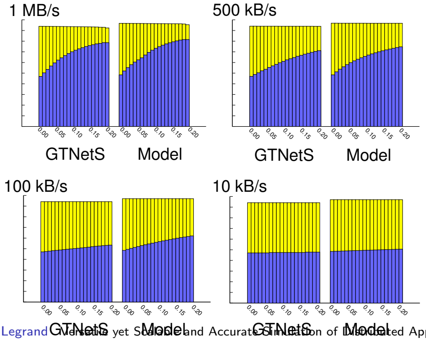



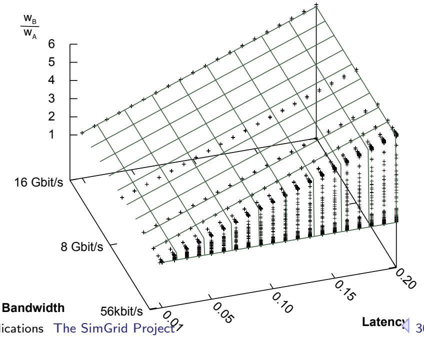
# Wanted Feature (3): RTT-unfairness

**Hypothesis:** **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \textbf{ where } \sout{RTT_i \approx \sum_{\text{flow i uses link j}} (L_j)} \text{ (naive model)}$$

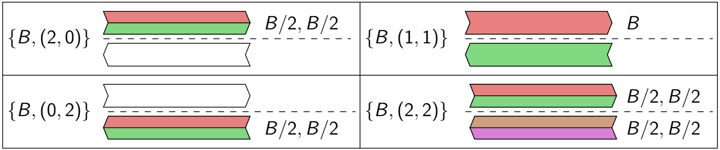
- Longer flows (higher latency) will receive slightly less bandwidth
- However, bandwidth also matters
- Again, instantiation improvement: $RTT_i \approx \sum_{\text{flow i uses link j}} \left( \frac{M}{B_j} + L_j \right)$



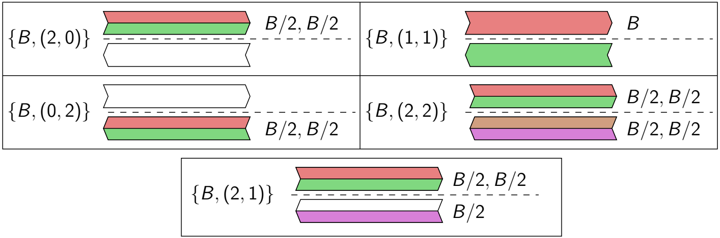
# Wanted Feature (4): Cross-Traffic Interference

Take two machines connected by a full-duplex ethernet link.



# Wanted Feature (4): Cross-Traffic Interference

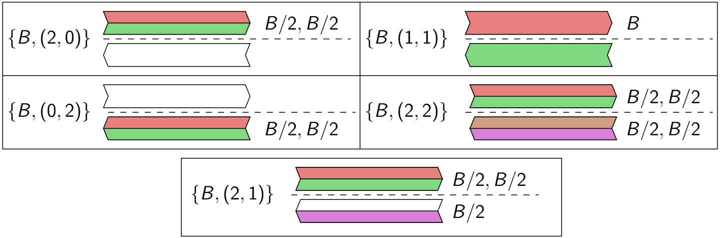
Take two machines connected by a full-duplex ethernet link.




This is a well-known phenomenon when you are using ADSL

# Wanted Feature (4): Cross-Traffic Interference

Take two machines connected by a full-duplex ethernet link.




This is a well-known phenomenon when you are using ADSL

**Burstiness at micro-scale severely impact macro-scale properties**

Modeling such burstiness is ongoing research and resorts to complex differential algebraic equations

Tang et al., *Window Flow Control: Macroscopic Properties from Microscopic Factors*, in INFOCOM 2008

# Wanted Features!

## Key characteristics of TCP

- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

## That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum$ log, $\sum$ arctan, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

## Invalidation studies: an endless quest?

# Wanted Features!

## Key characteristics of TCP

- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

## That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum$ log, $\sum$ arctan, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

## Invalidation studies: an endless quest?

# Wanted Features!

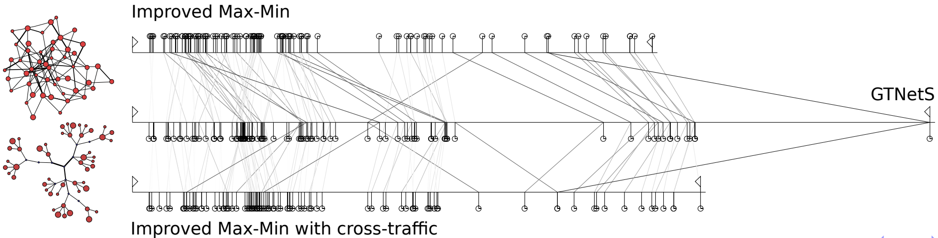
## Key characteristics of TCP

- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

## That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum$ log, $\sum$ arctan, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

## Invalidation studies: an endless quest?



# Wanted Features!

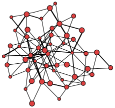
## Key characteristics of TCP

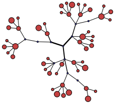
- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

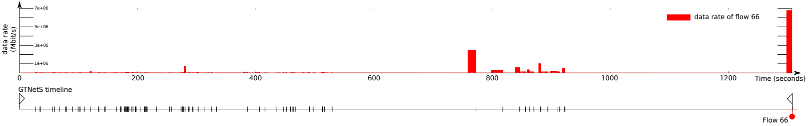
## That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum$ log, $\sum$ arctan, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

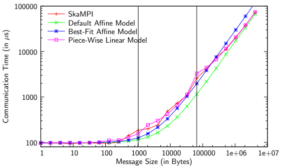
## Invalidation studies: an endless quest?







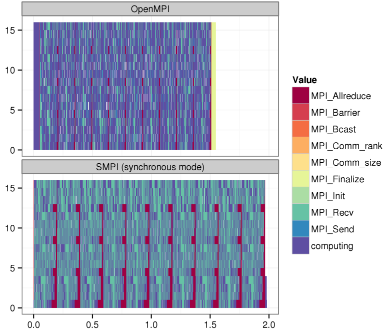
# Accuracy of MPI simulations




## Timings of each communication

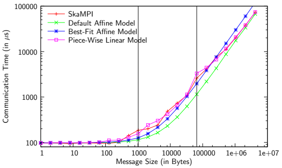
- $\lambda + size \times \tau$ not sufficient (TCP congestion)
- No affine fonction can match for all message sizes
- Piecewise affine model gives satisfying resultsœ
- Need to model communication/computation overlap

## Still a work in progress for complete MPI applications




- Sweep3D, OpenMPI, TCP, Gigabit Ethernet, 16 nodes
- Quite encouraging
- Take resource sharing into account
- Not only makespan comparison but also internal state distribution

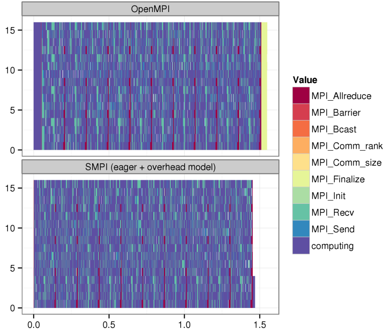
# Accuracy of MPI simulations




## Timings of each communication

- $\lambda + \mathit{size} \times \tau$ not sufficient (TCP congestion)
- No affine fonction can match for all message sizes
- Piecewise affine model gives satisfying resultsœ
- Need to model communication/computation overlap

## Still a work in progress for complete MPI applications




- Sweep3D, OpenMPI, TCP, Gigabit Ethernet, 16 nodes
- Quite encouraging
- Take resource sharing into account
- Not only makespan comparison but also internal state distribution

# Wrap up on network models

SotA: Models in most simulators are either simplistic, wrong or not assessed

- PeerSim: discrete time, application as automaton; GridSim: naive packet level
- OptorSim, GroudSim: documented as wrong on heterogeneous platforms
- *Validity evaluation:* tricky, requires meticulous attention & sound methodology

## SimGrid and the validation quest

**Fluid models can account** for TCP key characteristics

- slow-start
- flow-control limitation
- RTT-unfairness
- cross traffic interference

They are a very reasonable approximation for most LSDC systems

Yet, many people think they are too complex to scale.

Well, if you do things right, it's ok! ⌣̈

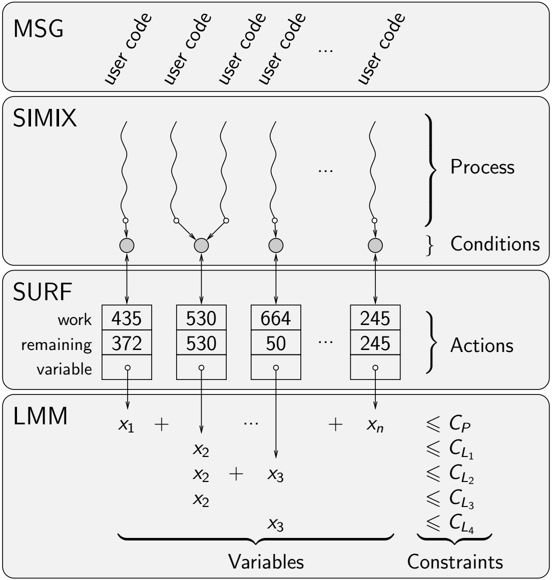
# SimGrid Internals in a Nutshell for Users

## 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 and NS3 wrapper
- Build your own (!)



# Outline

# A BOINC simulation using SimGrid

Volunteer are very unstable, which slows down simulation.

- Lazy Update + Efficient Future Event Set
- Traces Integration and Dichotomic Search

As a proof of structure, we coded a simplified BOINC architecture in about 800 lines and compared it to previous approaches.

BOINC client simulator the project sharings and deadline misses are very close to the ones observed with the BOINC client simulator.

- Surprisingly, our simulation is about 60 times faster. Who cares? It was already fast.
- More interesting: we can feed clients with SETI traces (e.g., from `http://fta.inria.fr`) and use the same code to simulate the whole system!

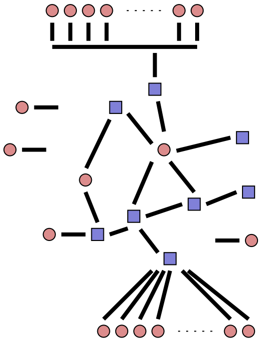
SimBA The authors reported the following performances on a P4 3GHz.

- P@H: 15 simulation days, 7810 workers = 107 minutes.
  We fed SimGrid with complex SETI traces: it takes less than 4 minutes!
- CHARMM: 8 simulation days, 5093 workers = 44 minutes.
  Simgrid result for a similar experiment: about 80 seconds.

But more importantly, BOINC can be studied as a whole (multiple clients, multiple projects, complex traces, realistic network models if needed).

# Scalable Platform Description

*N* nodes and *E* links



Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|

# Scalable Platform Description

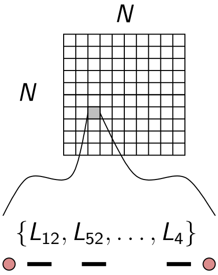
$N$ nodes and $E$ links

Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

Classical network representation

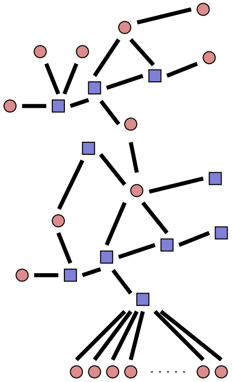
1. Flat representation
   5000 hosts doesn't fit in 4Gb!




| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Flat | $N^2$ | $N^2$ | $N^2$ | 1 |

# Scalable Platform Description

$N$ nodes and $E$ links



Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

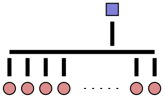
Classical network representation

1. Flat representation
   5000 hosts doesn't fit in 4Gb!
2. Graph representation assuming shortest path routing

| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Dijsktra | $N + E$ | $E + N \log N$ | $N + E$ | $E + N \log N$ |
| Floyd | $N + E$ | $N^2$ | $N^3$ | 1 |

# Scalable Platform Description

*N* nodes and *E* links

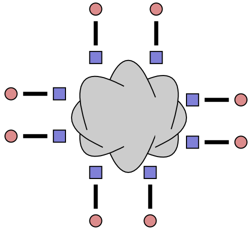


Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

Classical network representation

1. Flat representation
   5000 hosts doesn't fit in 4Gb!
2. Graph representation assuming shortest path routing
3. Special class of structures (star, cloud, ...)

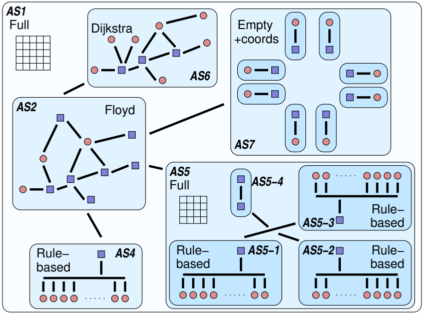


| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Star | 1 | $N$ | $N$ | 1 |
| Cloud | $N$ | $N$ | $N$ | 1 |

# SimGrid Platform Representation

Every such representation has drawbacks and advantages
Let's build on the fact that most networks are mostly hierarchical

1. Hierarchical organization in AS
   ⇝ cuts down complexity
   ⇝ recursive routing
2. Efficient representation of classical structures
3. Allow bypass at any level



# How big and how fast (1/3)? Grid

## Size of platform description file

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

# How big and how fast (1/3)? Grid

### Size of platform description file

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

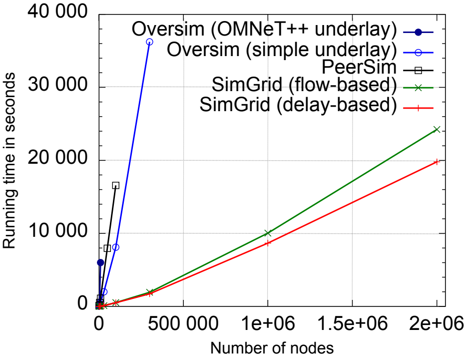
**Grid Scenario** a master distributes 500,000 fixed size jobs to 2,000 workers in a round-robin way

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

$\star$ 5.2Mb are used to represent the Grid 5000. Stack size not optimized (80KB/worker)

# How big and how fast (2/3)? P2P

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



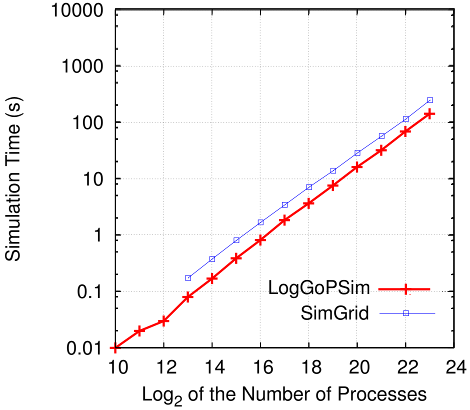

## Largest simulated scenario

| Simulator | size | time |
|---|---|---|
| OverSim (OMNeT++) | 10k | 1h40 |
| OverSim (simple) | 300k | 10h |
| PeerSim | 100k | 4h36 |
| SG (flow-based) | 10k<br>300k<br>2M* | 130s<br>32mn<br>6h23 |
| SG (delay-based) | 2M | 5h30 |

∗ 36GB = 18kB/ process (16kB for the stack)

- SimGrid is orders of magnitude more scalable than state-of-the-art P2P simulators
- Using the flow-based model incurs a limited (≈ 20%) slowdown, while simulation accuracy is improved

# How big and how fast (3/3)? HPC




Simulating a binomial broadcast:

- SimGrid is roughly 75% slower than LogGOPSim
- SimGrid is at least 20% more fat than LogGOPSim (15GB required for $2^{23}$ processors)

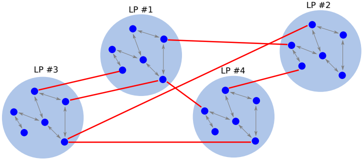
The genericity of SimGrid data structures comes at the cost of a slight overhead

This demonstrates that scalability does not necessarily comes at the price of realism (e.g., ignoring contention on the interconnect)

# Parallel P2P simulators: the dPeerSim attempt

## dPeerSim

- Parallel implementation of PeerSim/DES (not by PeerSim main authors)
- Classical parallelization: spreads the load over several Logical Processes (LP)



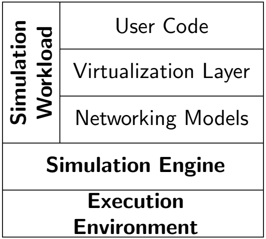

## Experimental Results

- Uses Chord as a standard workload: e.g. 320,000 nodes $\rightsquigarrow$ 320,000 requests
- The result are impressive at first glance
  - 4h10 using two Logical Processes: only 1h06 using 16 LPs
  - Speedup of 4 using 8 times more resources, that's really not bad at all
- But this is to be compared to sequential results
  - The same simulation takes 47 seconds in the original sequential PeerSim
  - (and 5 seconds using the precise network models of SimGrid in sequential)

# Parallel Simulation vs. Dist. Apps Simulators

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

Classical Parallel Simulation Schema
[Balakrishnan *et al*]



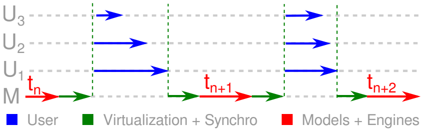

Layered View of
Dist. App. Simulators

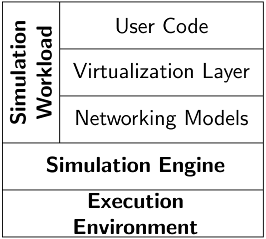
- The classical approach is to distribute the Simulation Engine entirely
- Hard issues: conservatives $\rightsquigarrow$ too few parallelism; optimistic $\rightsquigarrow$ roll back
- From our experience, most of the time is in so called "simulation workload"
  - User code executed as threads, that are scheduled according to simulation
  - The user code itself can reveal resource hungry: numerous / large processes

# Main Idea of this Work

## Split at Virtualization, not Simulation Engine

- Virtualization contains threads (user's stack)
- Engine & Models remains sequential




## Understanding the trade-off

- Sequential time: $\sum_{SR}(engine + model + virtu + user)$
- Classical schema: $\sum_{SR}\left(\max_{i \in LP}(engine_i + model_i + virtu_i + user_i) + proto\right)$
- Proposed schema: $\sum_{SR}\left(engine + model + \max_{i \in WT}(virtu_i + user_i) + sync\right)$
- Synchronization protocol expensive wrt the engine's load to be distributed

# Enabling Parallel Simulation of Dist.Apps

## Challenge: Allow User-Code to run Concurrently

- DES simulator full of shared data structures: how to avoid race conditions?
- Fine-locking would be difficult and inefficient; wouldn't avoid app-level races
  - $A$: *recv*, $B$: *send*, $C$: *send*; Which *send* matches the recv from $A$ in simulation?
  - Depends on execution order in host system $\rightsquigarrow$ simulation not reproducible...

# Enabling Parallel Simulation of Dist.Apps

## Challenge: Allow User-Code to run Concurrently

- DES simulator full of shared data structures: how to avoid race conditions?
- Fine-locking would be difficult and inefficient; wouldn't avoid app-level races
  - *A*: *recv*, *B*: *send*, *C*: *send*; Which *send* matches the recv from *A* in simulation?
  - Depends on execution order in host system $\rightsquigarrow$ simulation not reproducible...

## Solution: OS-inspired Separation Simulated Processes

- Mediate any interaction of processes with their environment, as in real OSes e.g. don't create a new process directly, but issue a *simcall* to request creation


$t \leftarrow 0$
$P_t \leftarrow P$
**while** $P_t \neq \emptyset$ **do**
    parallel_schedule($P_t$)
    handle_simcalls()
    $(t, events) \leftarrow$ models_solve()
    $P_t \leftarrow$ proc_to_wake($events$)
**end while**

# Enabling Parallel Simulation of Dist.Apps

## Challenge: Allow User-Code to run Concurrently

- DES simulator full of shared data structures: how to avoid race conditions?
- Fine-locking would be difficult and inefficient; wouldn't avoid app-level races
  - $A$: *recv*, $B$: *send*, $C$: *send*; Which *send* matches the recv from $A$ in simulation?
  - Depends on execution order in host system $\rightsquigarrow$ simulation not reproducible...

## Solution: OS-inspired Separation Simulated Processes

- Mediate any interaction of processes with their environment, as in real OSes e.g. don't create a new process directly, but issue a *simcall* to request creation

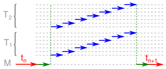

$t \leftarrow 0$
$P_t \leftarrow P$
**while** $P_t \neq \emptyset$ **do**
    parallel_schedule($P_t$)
    handle_simcalls()
    $(t, events) \leftarrow$ models_solve()
    $P_t \leftarrow$ proc_to_wake($events$)
**end while**

- Processes isolated from each others
  - Simcalls data locally stored
- Simcalls handled centrally once users blocked
  - Arbitrary fixed order for reproducibility

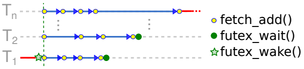
# Efficient Parallel Simulation

## Leveraging Multicores

- P2P involve millions of user processes, but dozens of cores at best
- Having millions of System threads is difficult (when possible)
- Co-routines (Unix ucontexts, Windows fibers): highly efficient but not parallel
- N:M model used: millions of coroutines executed on few threads




Logical View




Ideal Algorithm

## Reducing Synchronization Costs

- Inter-thread synchronization achieved through system calls (of real OS)
- Costs of syscalls are critical to performance $\leadsto$ save all possible syscalls
- Assembly reimplementation of ucontext: no syscall on context switch
- Synchronize only at scheduling round boundaries using futexes
- Dynamic load distribution: hardware fetch-and-add next process' index

# Microbenchmarking Synchronization Costs

Rq: P2P and Chord are ultra fine grain: this is thus a worst case scenario

## Comparing our user context containers

- pthreads hit a scalability limit by 32,000 processes (amount of semaphores)
- System contexts and ASM contexts have no hard limit (beside available RAM)

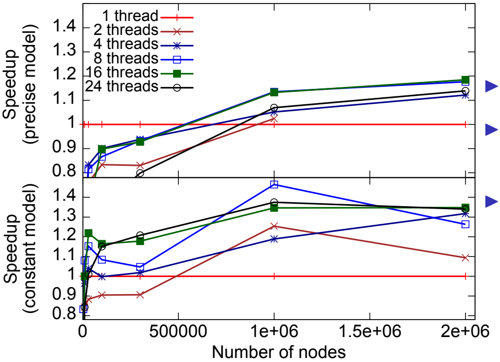
- pthreads are about 10 times slower than our own ASM contexts
- ASM contexts are about 20% faster than system ones
  (only difference: avoid any syscalls on user context switches)

## Measuring intrinsic synchronization costs

- Disabling parallelism at runtime: no noticeable performance change
- Enabling parallelism over 1 thread: 15% performance drop off

- Demonstrate the difficulty although the careful optimization

# Benefits of the Parallel Execution




- Speedup ($\frac{t_{seq}}{t_{par}}$): up to 45%
- More efficient with simple model:
  - Less work in engine + Amhdal law
- Speedup depends on thread amount
  - 8 threads (of 24 cores) often better
  - Synch costs remain hard to amortize
  - They depend on thread amount

## Parallel Efficiency ($\frac{speedup}{\#cores}$) for 2M nodes

| Model | 4 threads | 8 th. | 16 th. | 24 th. |
|---|---|---|---|---|
| Precise | 0.28 | 0.15 | 0.07 | 0.05 |
| Constant | 0.33 | 0.16 | 0.08 | 0.06 |

- Baaaaad efficiency results
- Remember, P2P and Chord: Worst case scenarios

Yet, first time that Chord's parallel simulation is faster than best known sequential

# 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 or Applications

- Packet-level simulators too slow for large scale studies
- Large amount of ad-hoc simulators, but discutable validity
- Coarse-grain modeling 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.

# 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 try to but ... (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 try to but ... (shame, shame). Why?

## Technical issues to tackle

- Archiving facilities, Versionning, Branch support, Dependencies management
- Workflows automating execution of test campaigns (`myexperiment.org`) and help sharing results (`manyeyes.alphaworks.ibm.com`)
- 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 try to but ... (shame, shame). Why?

## Technical issues to tackle

- Archiving facilities, Versionning, Branch support, Dependencies management
- Workflows automating execution of test campaigns (`myexperiment.org`) and help sharing results (`manyeyes.alphaworks.ibm.com`)
- 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?

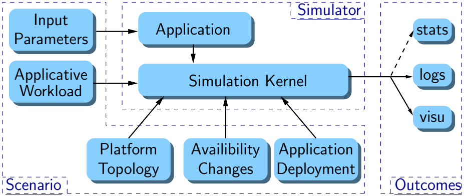
# Building Open Science Around the Simulator

## 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

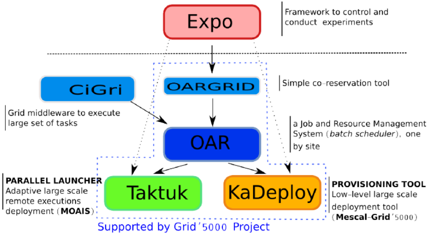



Figure from Olivier Richard

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

# 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 ☺

# 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)

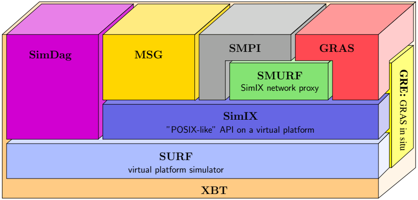
- Runs unmodified MPI code after recompilation (still partial implementation)

## 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)
- +100 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!**

# 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