**SIMGrid:** a Generic Framework for Large-Scale Distributed Experiments

Henri Casanova (Hawai’i University at Manoa, USA)
Arnaud Legrand (CNRS at Grenoble, France)
Martin Quinson (Nancy University, France)

UKSim 2008,
Cambridge, UK.
Large-Scale Distributed Systems Research

Large-scale distributed systems are in production today

- Grid platforms for "e-Science" applications
- Peer-to-peer file sharing
- Distributed volunteer computing
- Distributed gaming

Researchers study a broad area of systems

- Data lookup and caching algorithms
- Application scheduling algorithms
- Resource management and resource sharing strategies

They want to study several aspects of their system performance

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

**Main question:** comparing several solutions in relevant settings
Classical Experimental Methodologies

Analytical works?

- Some purely mathematical models exist
- Allow better understanding of principles (impossibility theorems)
- Theoretical results are difficult to achieve (without unrealistic assumptions)

⇒ Most published research in the area is experimental

Real-world experiments?

- Eminently believable to demonstrate the proposed approach applicability
- Very time and labor consuming; Reproducibility issues

⇒ Most published results rely on simulation or emulation

Simulation and emulation?

- Solve most issues of real-world experiments (fast, easy, unlimited and repeatable)
- Validation issue (amongst others)

⇒ Tools validity must be carefully assessed
Outline

- Introduction
- State of the Art
- SIMGrid Models
- SIMGrid User Interfaces
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion
### Some Existing Experimental Tools

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
</tbody>
</table>

- Large platforms: getting access is problematic, fixed experimental settings
### Some Existing Experimental Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
</tbody>
</table>

- **Large platforms:** getting access is problematic, fixed experimental settings
- **Virtualization:** no control over experimental settings
Some Existing Experimental Tools

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
</tbody>
</table>

- Large platforms: getting access is problematic, fixed experimental settings
- Virtualization: no control over experimental settings
- Emulation: hard to setup, can have high overheads
Some Existing Experimental Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
<td>&lt;1,000</td>
</tr>
<tr>
<td>SSFNet</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>&lt;100,000</td>
</tr>
<tr>
<td>GTNetS</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++</td>
<td>controlled</td>
<td>&lt;177,000</td>
</tr>
</tbody>
</table>

- Large platforms: getting access is problematic, fixed experimental settings
- Virtualization: no control over experimental settings
- Emulation: hard to setup, can have high overheads
- Packet-Level simulators: too network-centric (no CPU) and rather slow
### Some Existing Experimental Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
<td>&lt;1,000</td>
</tr>
<tr>
<td>SSFNet</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>&lt;100,000</td>
</tr>
<tr>
<td>GTNetS</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++</td>
<td>controlled</td>
<td>&lt;177,000</td>
</tr>
<tr>
<td>PlanetSim</td>
<td>-</td>
<td>-</td>
<td>cste time</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>100,000</td>
</tr>
<tr>
<td>PeerSim</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>state machine</td>
<td>Java</td>
<td>controlled</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

- **Large platforms**: getting access is problematic, fixed experimental settings
- **Virtualization**: no control over experimental settings
- **Emulation**: hard to setup, can have high overheads
- **Packet-Level simulators**: too network-centric (no CPU) and rather slow
- **P2P simulators**: great scalability, poor realism
## Some Existing Experimental Tools

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid’5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
<td>&lt;1,000</td>
</tr>
<tr>
<td>SSFNet</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>&lt;100,000</td>
</tr>
<tr>
<td>GTNetS</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++</td>
<td>controlled</td>
<td>&lt;177,000</td>
</tr>
<tr>
<td>PlanetSim</td>
<td>-</td>
<td>-</td>
<td>cste time</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>100,000</td>
</tr>
<tr>
<td>PeerSim</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>state machine</td>
<td>Java</td>
<td>controlled</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ChicSim</td>
<td>coarse d.e.</td>
<td>-</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>C</td>
<td>controlled</td>
<td>thousands</td>
</tr>
<tr>
<td>OptorSim</td>
<td>coarse d.e.</td>
<td>amount</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 100</td>
</tr>
<tr>
<td>GridSim</td>
<td>coarse d.e.</td>
<td>math</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Disk</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid’5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
<td>&lt;1,000</td>
</tr>
<tr>
<td>SSFNet</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>&lt;100,000</td>
</tr>
<tr>
<td>GTNetS</td>
<td>-</td>
<td>-</td>
<td>fine d.e.</td>
<td>coarse d.e.</td>
<td>C++</td>
<td>controlled</td>
<td>&lt;177,000</td>
</tr>
<tr>
<td>PlanetSim</td>
<td>-</td>
<td>-</td>
<td>cste time</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>100,000</td>
</tr>
<tr>
<td>PeerSim</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>state machine</td>
<td>Java</td>
<td>controlled</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ChicSim</td>
<td>coarse d.e.</td>
<td>-</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>C</td>
<td>controlled</td>
<td>thousands</td>
</tr>
<tr>
<td>OptorSim</td>
<td>coarse d.e.</td>
<td>amount</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 100</td>
</tr>
<tr>
<td>GridSim</td>
<td>coarse d.e.</td>
<td>math</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 100</td>
</tr>
<tr>
<td>SIMGrid</td>
<td>math/d.e.</td>
<td>(underway)</td>
<td>math/d.e.</td>
<td>d.e./emul</td>
<td>C or Java</td>
<td>controlled</td>
<td>few 10,000</td>
</tr>
</tbody>
</table>

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

Analytical Models proposed in literature

- Data streams modeled as **fluids in pipes**
Analytical Network Models

Analytical Models proposed in literature

- Data streams modeled as *fluids in pipes*

![Diagram of network flows and links](image)

Max-Min Fairness

- One of the possible way to compute the transfer rates ($\lambda_f$)
- Objective function: $\maximize \min_{f \in \mathcal{F}} (\lambda_f)$
- Equilibrium reached if unable to increase any rate without decreasing another
- Gives a fair share to everyone
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 $l$; $\lambda_f$: transfer rate of $f$.

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

\[
\begin{align*}
C_0 &= 1 & n_0 &= 1 \\
C_1 &= 1000 & n_1 &= 1 \\
C_2 &= 1000 & n_2 &= 2 \\
C_3 &= 1000 & n_3 &= 1 \\
C_4 &= 1000 & n_4 &= 1 \\
\end{align*}
\]
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 \( l \); \( \lambda_f \): transfer rate of \( f \).

\[
\begin{align*}
C_0 &= 0 & n_0 &= 0 \\
C_1 &= 1000 & n_1 &= 1 \\
C_2 &= 999 & n_2 &= 1 \\
C_3 &= 1000 & n_3 &= 1 \\
C_4 &= 999 & n_4 &= 0 \\
\lambda_1 &= \lambda_2 &= 1
\end{align*}
\]

- The limiting link is 0
- This fixes \( \lambda_2 = 1 \). Update the links
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 $l$;  $\lambda_f$: transfer rate of $f$.

<table>
<thead>
<tr>
<th>Link</th>
<th>Capacity ($C_l$)</th>
<th>Flows ($n_l$)</th>
<th>Transfer Rate ($\lambda_f$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>999</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>999</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- 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

$C_i$: capacity of link $i$; $n_i$: amount of flows using $i$; $\lambda_f$: transfer rate of $f$.

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

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

▷ The limiting link is 0
▷ This fixes $\lambda_2 = 1$. Update the links
▷ The limiting link is 2
▷ This fixes $\lambda_1 = 999$
SimGrid Models Evaluation: accuracy

Relative error of SimGrid over GTNetS on a dogbone topology

- Short messages: poor accuracy (no TCP slow-start yet)
- Reasonable Network Contention: very good! (error below 1%)
- Higher Network Contention: room for improvement (up to 100% on outliers)
# SimGrid Models Evaluation: speed

## 1Mb flows

<table>
<thead>
<tr>
<th># of flows</th>
<th>GTNetS Running time</th>
<th>GTNetS slowdown</th>
<th>SIMGrid Running time</th>
<th>SIMGrid slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.661s</td>
<td>0.856</td>
<td>0.002s</td>
<td>0.002</td>
</tr>
<tr>
<td>100</td>
<td>7.649s</td>
<td>7.468</td>
<td>0.137s</td>
<td>0.140</td>
</tr>
<tr>
<td>200</td>
<td>15.705s</td>
<td>11.515</td>
<td>0.536s</td>
<td>0.396</td>
</tr>
</tbody>
</table>

## 100Mb flows

<table>
<thead>
<tr>
<th># of flows</th>
<th>GTNetS Running time</th>
<th>GTNetS slowdown</th>
<th>SIMGrid Running time</th>
<th>SIMGrid slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>65s</td>
<td>0.92</td>
<td>0.001s</td>
<td>0.000002</td>
</tr>
<tr>
<td>100</td>
<td>753s</td>
<td>8.08</td>
<td>0.138s</td>
<td>0.00142</td>
</tr>
<tr>
<td>200</td>
<td>1562s</td>
<td>12.59</td>
<td>0.538s</td>
<td>0.00402</td>
</tr>
</tbody>
</table>

- GTNetS linear in number of flows and data size
- SIMGrid only linear in number of flows
SimGrid Models are Plugins

“--cfg=network_model” command line argument

- CM02 ➔ MaxMin fairness
- Vegas ➔ Vegas TCP fairness (Lagrange approach)
- Reno ➔ Reno TCP fairness (Lagrange approach)
- By default in SimGrid v3.3: CM02
- Example: ./my_simulator --cfg=network_model:Vegas

CPU sharing policy

- Default MaxMin is sufficient for most cases
- cpu_model:ptask_L07 ➔ model specific to parallel tasks

Want more?

- network_model:gtnets ➔ use Georgia Tech Network Simulator for network
  Accuracy of a packet-level network simulator without changing your code (!)
- Plug your own model in SimGrid!
  (usable as scientific instrument in TCP modeling field, too)
Outline

- Introduction
- State of the Art
- SimGrid Models
  - SimGrid User Interfaces
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
- Conclusion
# User-visible SimGrid Components

<table>
<thead>
<tr>
<th>SimDag</th>
<th>MSG</th>
<th>GRAS</th>
<th>AMOK</th>
<th>SMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework for DAGs of parallel tasks</td>
<td>Simple application-level simulator</td>
<td>Framework to develop distributed applications</td>
<td>toolbox</td>
<td>Library to run MPI applications on top of a virtual environment</td>
</tr>
</tbody>
</table>

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

## SimGrid user APIs

- **SimDag**: specify heuristics as DAG of (parallel) tasks
- **MSG**: specify heuristics as Concurrent Sequential Processes
  (Java bindings available)
- **GRAS**: develop real applications, studied and debugged in simulator
  **AMOK**: set of distributed tools (bandwidth measurement, failure detector, . . .)
- **SMPI**: simulate MPI codes *(still under development)*
# User-visible SimGrid Components

<table>
<thead>
<tr>
<th>SimDag</th>
<th>MSG</th>
<th>GRAS</th>
<th>AMOK</th>
<th>SMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework for DAGs of parallel tasks</td>
<td>Simple application-level simulator</td>
<td>Framework to develop distributed applications</td>
<td>Toolbox</td>
<td>Library to run MPI applications on top of a virtual environment</td>
</tr>
</tbody>
</table>

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

## SimGrid user APIs

- **SimDag**: specify heuristics as DAG of (parallel) tasks
- **MSG**: specify heuristics as Concurrent Sequential Processes (Java bindings available)
- **GRAS**: develop real applications, studied and debugged in simulator
  - **AMOK**: set of distributed tools (bandwidth measurement, *failure detector*, …)
- **SMPI**: simulate MPI codes (*still under development*)

## 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
Main functionalities

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

2. Schedule tasks on resources

3. Run the simulation (respecting precedences)
   ~ Compute the makespan

grounded experiments of half a dozen scientific publications
MSG: Heuristics for Concurrent Sequential Processes

(historical) Motivation

- Centralized scheduling does not scale
- SimDag 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
- **Channel**: mailbox number on an host (MPI tag)

Usage

- Was used for Grid Scheduling, Desktop Grid, P2P Systems, . . .
  (grounded ≈ 20 publications, not counting ours)
- Java bindings exist for the ones reluctant to C
GRAS (Grid Reality And Simulation)

Ease development of real distributed applications using a simulator

Framework for Rapid Development of Distributed Infrastructure

- Develop and tune on the simulator; Deploy *in situ* without modification
**GRAS (Grid Reality And Simulation)**

Ease development of real distributed applications using a simulator

Without GRAS

With GRAS

Framework for Rapid Development of Distributed Infrastructure

- Develop and tune on the simulator; Deploy *in situ* without modification
  
  How: One API, two implementations
**GRAS (Grid Reality And Simulation)**

Ease development of real distributed applications using a simulator

Framework for Rapid Development of Distributed Infrastructure

- **Develop and tune** on the simulator; **Deploy in situ** without modification

  **How:** One API, two implementations
GRAS (Grid Reality And Simulation)

Ease development of real distributed applications using a simulator

Framework for Rapid Development of Distributed Infrastructure

- Develop and tune on the simulator; Deploy \textit{in situ} without modification
  
  \textbf{How}: One API, two implementations

Efficient Grid Runtime Environment \((result = application \neq prototype)\)

- \textbf{Performance concern}: efficient communication of structured data
  
  \textbf{How}: Efficient wire protocol (avoid data conversion when possible)

- \textbf{Portability concern}: because of grid heterogeneity

\textbf{≈} Linux, Mac OSX, Windows, AIX, Solaris
Simulation Scalability

Implementation details

- Use of UNIX98 contexts when available
- No hard limit in libc or kernel (only memory)
- Ran 2,000,000 simulated processes (on a 16Gb host)

Comparing the Java and Native version

- Classical master/slaves example

<table>
<thead>
<tr>
<th># tasks</th>
<th>Native version</th>
<th>Java version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.7s</td>
<td>0.5s</td>
</tr>
<tr>
<td>10,000</td>
<td>1.7s</td>
<td>2.5s</td>
</tr>
<tr>
<td>100,000</td>
<td>9.6s</td>
<td>23s</td>
</tr>
<tr>
<td>1,000,000</td>
<td>96s</td>
<td>240s</td>
</tr>
</tbody>
</table>

- Performance linear to amount of task
- Difference: comparison of Java threads and ucontexts
Outline

- Introduction
- State of the Art
- SIMGrid Models
- SIMGrid User Interfaces
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion
Conclusions

Simulating Large-Scale Distributed Systems

- Packet-level simulators too slow for large scale studies
- Large amount of grid and P2P simulators, but discutable validity
- Coarse-grain modelization of TCP flows possible (cf. networking community)

SimGrid provides interesting models

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

SimGrid provides several user interfaces

- SimDag: Comparing Scheduling Heuristics for DAGs of (parallel) tasks
- MSG: Comparing Heuristics for Concurrent Sequential Processes
- GRAS: Developing and Debugging Real Applications
- Other ones coming: SMPI, BSP, OpenMP

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

- Used in over 50 research articles
- LGPL, 120,000 lines of code; Examples, docs and tutorials on the web page
Future work

- Go beyond memory limitation by partial parallelization
- Model-checking of GRAS applications
- Emulation solution is spirit of MicroGrid

http://simgrid.gforge.inria.fr/
Appendix
Network Models

Store and forward

- First idea, quite natural
- Pay price of link 1, then link 2
- Analogy to time from city to city
- Plainly wrong (data is packetized)

Wormhole Model

(used in GridSim and ChicSim)

- As slow as packet-level
- TCP congestion mechanism neglected
  \Rightarrow Poor accuracy
Side note: OptorSim 2.1 on Backbone

OptorSim (developed @CERN for Data-Grid)

- http://sourceforge.net/projects/optorsim
- One of the rare grid simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$

2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left( \frac{C_l}{n_l} \right)$
Side note: OptorSim 2.1 on Backbone

OptorSim (developed @CERN for Data-Grid)

- http://sourceforge.net/projects/optorsim
- One of the rare grid simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: \( \frac{C_l}{n_l} \)

2. For each flow, compute what it gets: \( \lambda_f = \min_{l \in f} \left( \frac{C_l}{n_l} \right) \)

\[
\begin{align*}
C_0 &= 1 & n_1 &= 1 & \text{share} = \\
C_1 &= 1000 & n_1 &= 1 & \text{share} = \\
C_2 &= 1000 & n_2 &= 2 & \text{share} = \\
C_3 &= 1000 & n_3 &= 1 & \text{share} = \\
C_4 &= 1000 & n_4 &= 1 & \text{share} = \\
\end{align*}
\]

\( \lambda_1 = \) \\
\( \lambda_2 = \)
Side note: OptorSim 2.1 on Backbone

OptorSim (developed @CERN for Data-Grid)

- http://sourceforge.net/projects/optorsim
- One of the rare grid simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: \( \frac{C_l}{n_l} \)

2. For each flow, compute what it gets: \( \lambda_f = \min_{l \in f} \left( \frac{C_l}{n_l} \right) \)

\[
\begin{align*}
C_0 &= 1 & n_1 &= 1 & \text{share} &= 1 \\
C_1 &= 1000 & n_1 &= 1 & \text{share} &= 1000 \\
C_2 &= 1000 & n_2 &= 2 & \text{share} &= 500 \\
C_3 &= 1000 & n_3 &= 1 & \text{share} &= 1000 \\
C_4 &= 1000 & n_4 &= 1 & \text{share} &= 1000 \\
\end{align*}
\]

\[
\begin{align*}
\lambda_1 &= \min(1000, 500, 1000) \\
\lambda_2 &= \min(1, 500, 1000)
\end{align*}
\]
Side note: OptorSim 2.1 on Backbone

OptorSim (developed @CERN for Data-Grid)

▶ http://sourceforge.net/projects/optorsim
▶ One of the rare grid simulators not using wormhole

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: \( \frac{C_l}{n_l} \)

2. For each flow, compute what it gets: \( \lambda_f = \min_{l \in f} \left( \frac{C_l}{n_l} \right) \)

\[
\begin{align*}
C_0 &= 1 & n_1 &= 1 & \text{share} &= 1 \\
C_1 &= 1000 & n_1 &= 1 & \text{share} &= 1000 \\
C_2 &= 1000 & n_2 &= 2 & \text{share} &= 500 \\
C_3 &= 1000 & n_3 &= 1 & \text{share} &= 1000 \\
C_4 &= 1000 & n_4 &= 1 & \text{share} &= 1000 \\
\end{align*}
\]

\[
\begin{align*}
\lambda_1 &= \min(1000, 500, 1000) = 500!! \\
\lambda_2 &= \min(1, 500, 1000) = 1
\end{align*}
\]

Listed as “unwanted feature” in the README file...
**Simulation Main Loop**

**Data:** set of resources with working rate
Simulation Main Loop

**Data:** set of resources with working rate

1. Some **actions** get created and assigned to resources
**Simulation Main Loop**

**Data:** set of resources with working rate

1. Some **actions** get created and assigned to resources
2. **Compute share** of everyone (resource sharing algorithms)
Simulation Main Loop

**Data:** set of resources with working rate

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

Data: set of resources with working rate

1. Some actions get created 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
Simulation Main Loop

**Data:** set of resources with working rate

1. Some **actions** get created 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

---

**Diagram:**
- Simulated time

---

Casanova, Legrand, Quinson

**SimGrid:** a Generic Framework for Large-Scale Distributed Experiments

UKSim’08, Cambridge.
Data: set of resources with working rate

1. Some actions get created 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
Simulation Main Loop

Data: set of resources with working rate
1. Some actions get created 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