SimGRID: a Generic Framework for Large-Scale Distributed Experiments

Martin Quinson (LORIA–Nancy University, France)
Arnaud Legrand (CNRS, Grenoble University, France)
Frédéric Suter (CNRS, IN2P3, Lyon, France)
Henri Casanova (Hawai’i University at Manoa, USA)
Introduction

Context: Research on Large-Scale Distributed Systems

- Systems already in use, even if not fully understood
- Researchers need to **assess and compare** solutions (algorithms, applications, etc.)

Experimental Methodologies

- **Analytical Work** difficult without unrealistic assumptions
- **Real-world Experiments**
  - 😊 Probably less experimental bias; ☹️ Time/labor consuming; Reproducibility?
- **Simulation/Emulation**
  - 😊 Fast, Easy, Unlimited, Repeatable; ☹️ Validation?
Introduction

Context: Research on Large-Scale Distributed Systems
- Systems already in use, even if not fully understood
- Researchers need to assess and compare solutions (algorithms, applications, etc.)

Experimental Methodologies
- Analytical Work difficult without unrealistic assumptions
- Real-world Experiments
  - 😊 Probably less experimental bias; 😞 Time/labor consuming; Reproducibility?
- Simulation/Emulation
  - 😊 Fast, Easy, Unlimited, Repeatable; 😞 Validation?

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 (work reviewing)

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
The SimGrid Project (Hawai`i, Grenoble, Nancy)

History

- Created just like other home-made simulators (only a bit earlier ;) for HPC
- Original goal: scheduling research $\rightarrow$ need for speed (users do parameter sweep)
- HPC quality criteria: makespan $\rightarrow$ accuracy not negligible

SimGrid in a Nutshell

- SimGrid is 10 years old: we explored several architectures, models, etc
- Many genericity hooks: modular, multi-API, multi-model $\rightarrow$ multi-community?

- Current work: pushing the scalability limits
- Some people study Desktop Grids with it
- We think it could be used in P2P too

Let’s try to convince you of that
Agenda

- Introduction
- SimGRID Overview
- Simulation Models
- Accuracy Assessment
- Scalability Assessment
- Conclusion
User-visible SimGrid Components

<table>
<thead>
<tr>
<th>SimDag</th>
<th>MSG</th>
<th>GRAS</th>
<th>AMOK toolbox</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

- **Specialized APIs:** Designed for a specific community, genericity not a goal
  - **SimDag:** model applications as DAG of (parallel) tasks
  - **SMPI:** simulate MPI codes

- **Generic APIs:** allow to express Concurrent Sequential Processes (CSP)
  - **MSG:** study heuristics, get quickly some performance evaluation charts
  - **GRAS:** develop real applications, studied and debugged in simulator
  - (+**XBT:** grounding toolbox easing C coding)

**Argh, you really expect me to code in C?!**

- Java bindings to MSG exist, other are planned (Python, C++, SimDag)
- Some bad sides of C avoided: feature-rich toolbox w/o dependency, portable
SimGrid Usage Workflow: the MSG example (1/2)

1. Write the Code of your Agents

```c
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
<?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
<?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

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

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

    MSG_main(); /* Launch the simulation */

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

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

Executive summary, but representative

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

- Introduction
- SimGRID Overview
- Simulation Models
- Accuracy Assessment
- Scalability Assessment
- Conclusion
Under the Hood: Simulation Models

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

Modeling Single-Hop Networks
► Simplistic: $T = \lambda + \frac{size}{\beta}$; Better: use $\beta' = min(\beta, \frac{W_{max}}{RTT})$ (TCP windowing)
Under the Hood: Simulation Models

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

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

Modeling Multi-Hop Networks
- Simplistic Models: Store & Forward or Wormhole

Easy to implement; Not realistic

(TCP Congestion omitted)
Under the Hood: Simulation Models

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

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

Modeling Multi-Hop Networks
- Simplistic Models: Store & Forward or Wormhole
  - Easy to implement; Not realistic (TCP Congestion omitted)
  - NS2 and other packet-level study the path of each and every network packet
    - Realism commonly accepted; Slow
Under the Hood: Simulation Models

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

Modeling Single-Hop Networks
▶ Simplistic: $T = \lambda + \frac{\text{size}}{\beta}$; Better: use $\beta' = \min(\beta, \frac{W_{\max}}{RTT})$ (TCP windowing)

Modeling Multi-Hop Networks
▶ Simplistic Models: Store & Forward or Wormhole

Easy to implement; Not realistic
▶ NS2 and other packet-level study the path of each and every network packet
Realism commonly accepted; Sloooooow
▶ Fluid Models: Data streams $\approx$ fluids in pipes
Fast, Rather well studied; Resource sharing; Would you trust that?
Agenda

- Introduction
- SimGRID Overview
- Simulation Models
- Accuracy Assessment
- Scalability Assessment
- Conclusion
Validation experiments on a single link

Experimental settings

- Compute achieved bandwidth as function of $S$
- Fixed $L=10\text{ms}$ and $B=100\text{MB/s}$

Evaluation Results

- Packet-level tools don’t completely agree
- SSFNet TCP_FAST_INTERVAL bad default
- GTNetS is equally distant from others
- Old SimGrid model omitted slow start effects
Validation experiments on a single link

Experimental settings

- Compute achieved bandwidth as function of $S$
- Fixed $L=10\text{ms}$ and $B=100\text{MB/s}$

Evaluation Results

- Packet-level tools don’t completely agree
- SSFNet TCP_FAST_INTERVAL bad default
- GTNetS is equally distant from others
- Old SimGrid model omitted slow start effects
- Statistical analysis of GTNetS slow-start
- Better instantiation of MaxMin model
  $\beta'' \sim 0.92 \times \beta'; \ \lambda \sim 10.4 \times \lambda$
- Resulting validity range quite acceptable

| $S$    | $|\varepsilon|$ | $|\varepsilon_{\text{max}}|$ |
|--------|-----------------|-----------------|
| $S < 100\text{KB}$ | $\approx 12\%$   | $\approx 162\%$       |
| $S > 100\text{KB}$ | $\approx 1\%$    | $\approx 6\%$         |
Validation experiments on random platforms

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

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

Maybe the error is not SimGrid’s

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

$\Rightarrow$ We’re adding random jitter to continue SimGRID validation
So, what is the model used in SimGrid?

"--cfg=network_model" command line argument

- **CM02, LV08** $\rightsuccsim$ MaxMin fairness (give a fair share to everyone)
- **Vegas** $\rightsuccsim$ Vegas TCP fairness (Lagrangian approach)
- **Reno** $\rightsuccsim$ Reno TCP fairness (Lagrangian approach)
- **By default:** LV08
- **Example:** 

```
./my_simulator --cfg=network_model:Vegas
```

CPU sharing policy

- Default MaxMin is sufficient for most cases
- `cpu_model:ptask_L07` $\rightsuccsim$ model specific to parallel tasks

Want more?

- `network_model:gtnets` $\rightsuccsim$ use Georgia Tech Network Simulator for network
  Accuracy of a packet-level network simulator without changing your code (!)
- Plug your own model in SimGrid!!
- Other models are currently cooking (constant time, last-mile, etc.)
Agenda

- Introduction
- SimGRID Overview
- Simulation Models
- Accuracy Assessment
- Scalability Assessment
- Conclusion
Simulation speed

200-nodes/200-flows network sending 1MB each

<table>
<thead>
<tr>
<th># of flows</th>
<th>Simulation time</th>
<th>SimGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.661s</td>
<td>0.856</td>
</tr>
<tr>
<td>100</td>
<td>7.649s</td>
<td>7.468</td>
</tr>
<tr>
<td>200</td>
<td>15.705s</td>
<td>11.515</td>
</tr>
</tbody>
</table>

200-nodes/200-flows network sending 100MB each

<table>
<thead>
<tr>
<th># of flows</th>
<th>Simulation time</th>
<th>SimGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>65s</td>
<td>0.92</td>
</tr>
<tr>
<td>100</td>
<td>753s</td>
<td>8.08</td>
</tr>
<tr>
<td>200</td>
<td>1562s</td>
<td>12.59</td>
</tr>
</tbody>
</table>

Conclusion

- GTNetS execution time linear in both data size and #flows
- SimGrid only depends on #flows
- (plus, GTNetS clearly outperforms NS2)
## Application-Level Benchmarks

### Master/Workers on amd64 with 4Gb

<table>
<thead>
<tr>
<th>#tasks</th>
<th>Context mechanism</th>
<th>#Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>----------</td>
</tr>
<tr>
<td>1,000</td>
<td>ucontext</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>pthread</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>0.41</td>
</tr>
<tr>
<td>10,000</td>
<td>ucontext</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>pthread</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>1.6</td>
</tr>
<tr>
<td>100,000</td>
<td>ucontext</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>pthread</td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>14.</td>
</tr>
<tr>
<td>1,000,000</td>
<td>ucontext</td>
<td>36.</td>
</tr>
<tr>
<td></td>
<td>pthread</td>
<td>42.</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>121.</td>
</tr>
</tbody>
</table>

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

### Extensibility with UNIX contexts

<table>
<thead>
<tr>
<th>#tasks</th>
<th>Stack size</th>
<th>#Workers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>25,000</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>1,000</td>
<td>128Kb</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>0.5</td>
</tr>
<tr>
<td>10,000</td>
<td>128Kb</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>0.8</td>
</tr>
<tr>
<td>100,000</td>
<td>128Kb</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>3.7</td>
</tr>
<tr>
<td>1,000,000</td>
<td>128Kb</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>33</td>
</tr>
<tr>
<td>5,000,000</td>
<td>128Kb</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>161</td>
</tr>
</tbody>
</table>

†: out of memory

### Scalability limit of GridSim

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

⇒ at most 10,922 user processes
## Application-Level Benchmarks

### Master/Workers on amd64 with 4Gb

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

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

- These results are old already (before the summer ;)
- v3.3.3 is 30% faster
- v3.3.4 → lazy evaluation

### Extensibility with UNIX contextes

<table>
<thead>
<tr>
<th>#tasks</th>
<th>Stack size</th>
<th>25,000</th>
<th>50,000</th>
<th>100,000</th>
<th>200,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>128Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td>10,000</td>
<td>128Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td>100,000</td>
<td>128Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td>1,000,000</td>
<td>128Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td>5,000,000</td>
<td>128Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
<tr>
<td></td>
<td>12Kb</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
<td>⌧</td>
</tr>
</tbody>
</table>

†: out of memory

### Scalability limit of GridSim

- 1 user process = 3 java threads (code, input, output)
- System limit = 32k threads
  ⇒ at most 10,922 user processes
Agenda

- Introduction
- SimGRID Overview
- Simulation Models
- Accuracy Assessment
- Scalability Assessment
- Conclusion
Conclusion

SimGRID is not P2P specific
► Initially: HPC community; already used in Desktop Grids

SimGRID could Help your Research anyway
► Provides Interesting Models: fast and shown accurate
  ► When chasing SimGRID accuracy limits, we found packet-level ones
  ► 30,000 requests/sec (and counting) in Master/Workers classical example
► Is Generic: multi-models, but also several user interfaces provided
► Is Configurable: Platform, Deployment, Workload and Code not intermixed
► Allows live deployments with GRAS (performance comparable to MPI)
► Enjoys a solid user community: 130 members on -user; grounded >40 papers

SimGRID is not perfect
► Learning curve harder: mainly C even if Java bindings exist
► Few associated tools: No GUI, no visualization, poor statistics (but a generator)
► No stock implementation
Conclusion

SimGrid is not P2P specific
- Initially: HPC community; already used in Desktop Grids

SimGrid could Help your Research anyway
- Provides Interesting Models: fast and shown accurate
  - When chasing SimGrid accuracy limits, we found packet-level ones
  - 30,000 requests/sec (and counting) in Master/Workers classical example
- Is Generic: multi-models, but also several user interfaces provided
- Is Configurable: Platform, Deployment, Workload and Code not intermixed
- Allows live deployments with GRAS (performance comparable to MPI)
- Enjoys a solid user community: 130 members on -user; grounded >40 papers

SimGrid is not perfect
- Learning curve harder: mainly C even if Java bindings exist
- Few associated tools: No GUI, no visualization, poor statistics (but a generator)
- No stock implementation

It’s a very active research project
- Ultra-Scalable Simulation with SimGrid: 3 years grant (1M$, 7 labs, 25 people)
  Plus other smaller grants ongoing or under evaluation
- Big Plans: Model-Checking; Emulation solution (plus usability improvement)