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)

ANR 08 SEGI 022

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

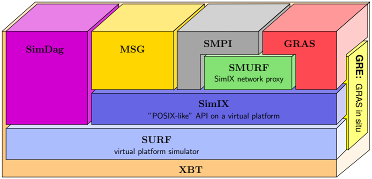
The SimGrid Project (Hawai'i, Grenoble, Nancy)

History

- Created just like other home-made simulators (only a bit earlier ;) for HPC
- Original goal: scheduling research $\rightsquigarrow$ need for speed (users do parameter sweep)
- HPC quality criteria: makespan $\rightsquigarrow$ 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 $\rightsquigarrow$ multi-community?



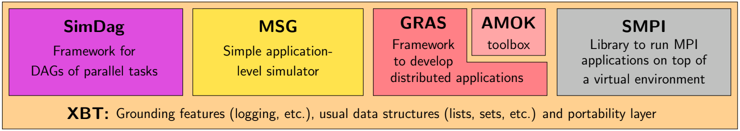
- Current work: pushing the scalability limits
- Some people study Desktop Grids with it
- We think it could be used in P2P too

Let's try to convince you of that

Agenda

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

User-visible SimGrid Components



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 planed (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

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

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

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

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

2. Describe your Experiment

XML Platform File

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

XML Deployment File

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

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

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

3. Glue things together

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

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

  MSG_main(); /* Launch the simulation */

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

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

Executive summary, but representative

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

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

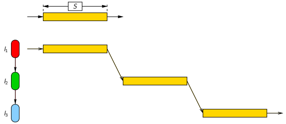
- Resource delivers *pow* flop / sec; task require *size* flop $\Rightarrow$ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

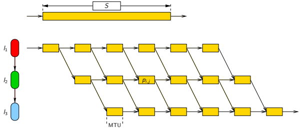
Modeling Single-Hop Networks

- 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)*

Under the Hood: Simulation Models

Modeling CPU

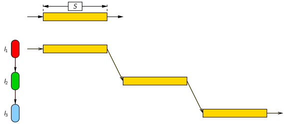
- Resource delivers *pow* flop / sec; task require *size* flop ⇒ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

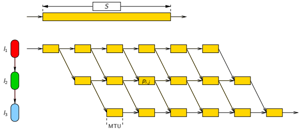
Modeling Single-Hop Networks

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

Modeling Multi-Hop Networks

- Simplistic Models: Store & Forward or Wormhole





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

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

☺ Realism commonly accepted; ☹ Sloooooow

Under the Hood: Simulation Models

Modeling CPU

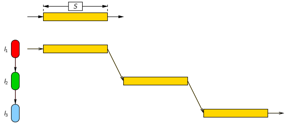
- Resource delivers *pow* flop / sec; task require *size* flop $\Rightarrow$ lasts $\frac{size}{pow}$ sec
- Simple (simplistic?) but more accurate become quickly intractable

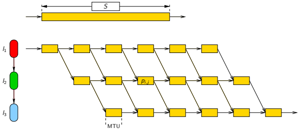
Modeling Single-Hop Networks

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

Modeling Multi-Hop Networks

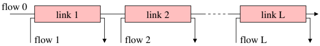
- Simplistic Models: Store & Forward or Wormhole





  ☺ Easy to implement; ☹ Not realistic *(TCP Congestion omitted)*
- NS2 and other packet-level study the path of each and every network packet

  ☺ Realism commonly accepted; ☹ Sloooooow



- Fluid Models: Data streams $\approx$ fluids in pipes

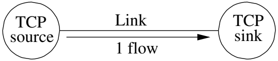
  ☺ Fast, Rather well studied; ☹ Resource sharing; Would you trust that?

Agenda

- Introduction
- SimGrid Overview
- Simulation Models
- **Accuracy Assessment**
- Scalability Assessment
- Conclusion

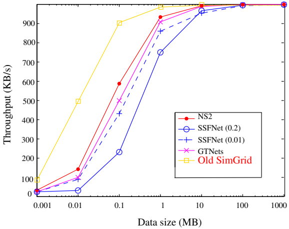
Validation experiments on a single link

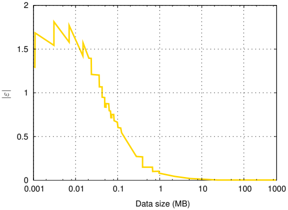
Experimental settings



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

Evaluation Results



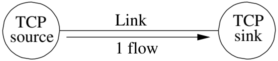


- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default
- GTNetS is equally distant from others

- Old SimGrid model omitted slow start effects

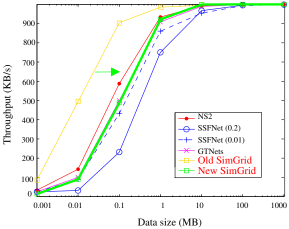
Validation experiments on a single link

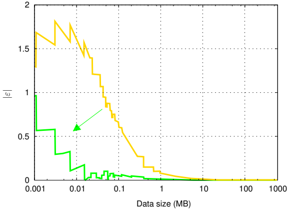
Experimental settings



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

Evaluation Results





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

- Old SimGrid model omitted slow start effects

$\Rightarrow$ Statistical analysis of GTNetS slow-start

$\rightsquigarrow$ Better instantiation of MaxMin model

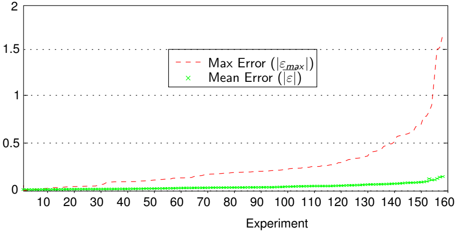
$\beta'' \rightsquigarrow .92 \times \beta'$; $\lambda \rightsquigarrow 10.4 \times \lambda$

- Resulting validity range quite acceptable

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

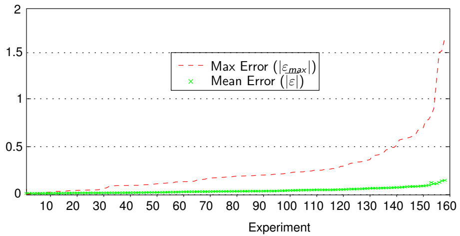
Validation experiments on random platforms

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



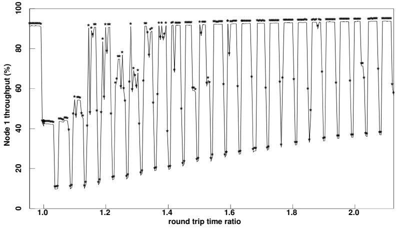
Validation experiments on random platforms

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



Maybe the error is not SimGrid's

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



$\rightsquigarrow$ 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` $\leadsto$ MaxMin fairness (give a fair share to everyone)
- `Vegas` $\leadsto$ Vegas TCP fairness (Lagrangian approach)
- `Reno` $\leadsto$ 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` $\leadsto$ model specific to parallel tasks

Want more?

- `network_model:gtnets` $\leadsto$ 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

Simulation speed

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

	GTNetS		SimGrid	
# of flows	Simulation time	$\frac{simulation}{simulated}$	Simulation time	$\frac{simulation}{simulated}$
10	0.661s	0.856	0.002s	0.002
100	7.649s	7.468	0.137s	0.140
200	15.705s	11.515	0.536s	0.396

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

	GTNetS		SimGrid	
# of flows	Simulation time	$\frac{simulation}{simulated}$	Simulation time	$\frac{simulation}{simulated}$
10	65s	0.92	0.001s	0.00002
100	753s	8.08	0.138s	0.00142
200	1562s	12.59	0.538s	0.00402

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

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

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

Extensibility with UNIX contextes

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

Scalability limit of GridSim

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

$\Rightarrow$ at most 10,922 user processes

†: out of memory

Application-Level Benchmarks

Master/Workers on amd64 with 4Gb

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

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

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

Extensibility with UNIX contextes

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

Scalability limit of GridSim

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

$\Rightarrow$ at most 10,922 user processes

†: out of memory

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)