Simulation Of Next Generation Systems
Computational Science of Distributed Systems

Martin Quinson

November 20, 2012

Our Scientific Objects: Distributed Systems

Cloud Computing

- Large infrastructures underlying commercial Internet (eBay, Amazon, Google)
- Main issues: Optimize costs; Keep up with the flash crowds load

P2P Systems

- Exploit resources at network edges (storage, CPU, human presence)
- Main issues: Churn and Resilience; Network locality; Anonymity

Scientific Computing: High Performance Computing / Computational Grids

- Infrastructure underlying *Computational science*: Massive / Federated systems
- Main issues: Have the world's biggest one / compatibility, trust, accountability

Systems already in use, but characteristics hard to assess

- Correction: absence of crash, race conditions, deadlocks and other defects
- Performance: makespan, economics, energy, ... ← main context of SimGrid

Assessing Distributed Applications

Correction Study $\leadsto$ Formal Methods

- Tests: Unable to provide definitive answers

Performance Study $\leadsto$ Experimentation

- Maths: Often not sufficient to fully understand these systems

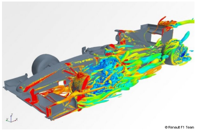
Assessing Distributed Applications

Correction Study $\rightsquigarrow$ Formal Methods

- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study $\rightsquigarrow$ Experimentation

- Maths: Often not sufficient to fully understand these systems



- Experimental Facilities: Real applications on Real platform *(in vivo)*

- Simulation: Prototypes of applications on system's Models *(in silico)*

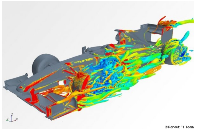
Assessing Distributed Applications

Correction Study $\rightsquigarrow$ Formal Methods

- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study $\rightsquigarrow$ Experimentation

- Maths: Often not sufficient to fully understand these systems



- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Emulation: Real applications on Synthetic platforms *(in vitro)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

Why Simulating Distributed Systems??

Why are Theoretical Studies not enough?

- Computers are Artificial Artifacts
- But computer systems present an Unprecedented Complexity
  - Heterogeneous components, Dynamic and Complex platforms
  - Numerous: milions of cores expected within the decade (ExaScale)
  - Large: Linux kernel only is 15M lines – over 10 times Encyclopedia Britanica

Toward a Computational Science of Distributed Computer Systems

- Empirically consider Distributed Systems as "Natural" Objects
- Other sciences routinely use computers to understand complex systems

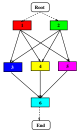
Claim: Simulation is both sound and convenient

- Less simplistic than proposed theoretical models
- Easier and faster than experimental platforms

Simulating Distributed Systems

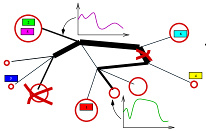
Big Idea: Simulation is the fastest path from idea to scientific results

Idea to test



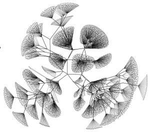
$+$

Experimental setup



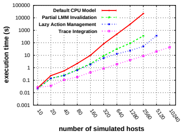
$+$

Simulation Model



$\Rightarrow$

Scientific results



Comfort to the user

- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, don't fiddle with technical subtleties (yet)

Challenges for the tools

- Validity: Get realistic results (controlled experimental bias)
- Scalability: Simulate *fast enough* problems *big enough*
- Associated tools: campaign mgmt, result analysis, settings generation, ...
- Applicability: If it doesn't simulate what is important to the user, it's void

Computational Science of Distributed Systems?

Requirements for a Scientific Approach

- Reproducible results: read a paper, reproduce the results and improve
- Standard tools that Grad students can learn quickly

Current practice in the field is quite different

- Experimental settings not detailed enough in literature
- Many short-lived simulators; few sound and established tools

	Domain	**CPU**	**Disk**	**Network**	**Application**	**Scale**
OptorSim	(Data)Grid	Analytic	Amount.	(buggy) Analytic	Programmatic	1,000
GridSim CloudSim	Grid Cloud	Analytic	Analytic	(buggy) wormhole (buggy) Analytic	Programmatic	1,000
OverSim	P2P	None	None	Euclidian or Pkt-lvl	Programmatic	100,000
PeerSim	P2P	None	None	Constant time	State machine	1,000,000
SimGrid	Grid, VC, P2P, HPC, cloud, . . .	Analytic	Amount	Flow, Cste-time or Packet-level (NS3)	Program, Trace or Emulation	1,000,000

SimGrid: Versatile Simulator of Distributed Apps

Toward **Computational Science** of Large-Scale Distributed Systems

Scientific Instrument

- Versatile: Grid, P2P, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Open: Grounded +100 papers; 100 members on simgrid-user@; LGPL

Scientific Object (and lab)

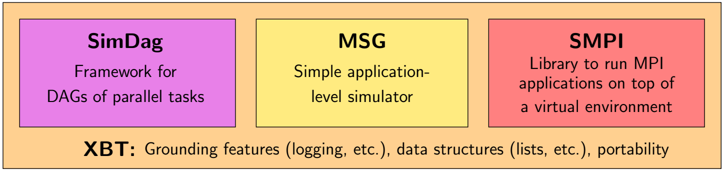
- Allows comparison of network models on non-trivial applications
- Experimental Model-Checker; full Emulator under way

Scientific Project since 12 years

- Initially a toolbox to factorize code between PhD students, in 1999
- Soon a collaboration Loria / Inria Rhône-Alpes / CCI-N2P3 / U. Hawaii
- Funding and support from INRIA since 2002
- Funding from French ANR: USS SimGrid (08-11) and SONGS (12-16)

Quick Overview of Internals Organization

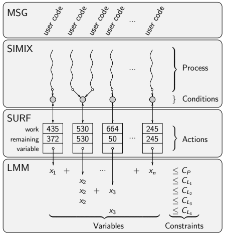
User-visible SimGrid Components


SimDag
Framework for DAGs of parallel tasks
MSG
Simple application-level simulator
SMPI
Library to run MPI applications on top of a virtual environment
XBT: Grounding features (logging, etc.), data structures (lists, etc.), portability

- MSG: heuristics as Concurrent Sequential Processes (Java/Ruby/Lua bindings)
- SimDag: heuristics as DAG of (parallel) tasks
- SMPI: simulate real applications written using MPI

SimGrid is Strictly Layered internaly

- MSG: User-friendly syntaxic sugar
- Simix: Processes, synchro (SimPosix)
- SURF: Resources usage interface
- Models: Action completion computation



Simulation Validity

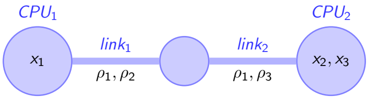
SotA: Models in most simulators are either simplistic, wrong or not assessed

- PeerSim: discrete time, application as automaton;
- GridSim: naive packet level or buggy flow sharing
- OptorSim, GroudSim: documented as wrong on heterogeneous platforms

SimGrid provides several Network Models

- Fast flow-based model, toward realism and speed (by default)
  Accounts for Contention, Slow-start, TCP congestion, Cross-traffic effects
- Constant time: A bit faster, but no hope of realism
- Coordinate-based: Easier to instantiate in P2P scenarios
- Packet-level: NS3 bindings
- Controlled by command line switches (exact comparison on a given application)

Max-Min Fairness between Network Flows



$$x_1 \leq Power\_CPU_1 \quad (1a)$$
$$x_2 + x_3 \leq Power\_CPU_2 \quad (1b)$$
$$\rho_1 + \rho_2 \leq Power\_link_1 \quad (1c)$$
$$\rho_1 + \rho_3 \leq Power\_link_2 \quad (1d)$$

Computing the sharing between flows

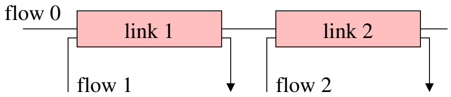
- Objective function: maximize $\min_{f \in \mathcal{F}}(\rho_f)$ [Massoulié & Roberts 2003]
- Equilibrium: increasing any ρ_f decreases a ρ'_f (with $\rho_f > \rho'_f$)
- (actually, that's a simplification of our real objective function)

Efficient Algorithm

1. Search for the bottleneck link l so that: $\frac{C_l}{n_l} = min\left\{\frac{C_k}{n_k},\ k \in \mathcal{L}\right\}$
2. This determines any flow f on this link: $\rho_f = \frac{C_l}{n_l}$
3. Update all n_l and C_l to remove these flows; Loop until all ρ_f are fixed

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = C \qquad n_1 = 2$

$C_2 = C \qquad n_2 = 2$

$\rho_0 =$

$\rho_1 =$

$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

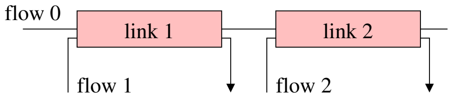
- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

Max-Min Fairness Example

Homogeneous Linear Network



$$C_1 = C \qquad n_1 = 2$$
$$C_2 = C \qquad n_2 = 2$$

$$\rho_0 = C/2$$
$$\rho_1 = C/2$$
$$\rho_2 =$$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

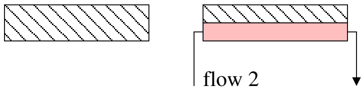
- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = 0 \qquad n_1 = 0$
$C_2 = C/2 \qquad n_2 = 1$

$\rho_0 = C/2$
$\rho_1 = C/2$
$\rho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

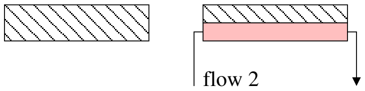
- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

Max-Min Fairness Example

Homogeneous Linear Network



$C_1 = 0 \qquad n_1 = 0$

$C_2 = 0 \qquad n_2 = 0$

$\rho_0 = C/2$

$\rho_1 = C/2$

$\rho_2 = C/2$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\rho_0 = C/2$ and $\rho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\rho_1 = C/2$.
- We are done computing the bandwidths ρ_i

Efficient Implementation

- Lazy updates, Trace integration, preserving Cache locality

Validity of GridSim and CloudSim

"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."

GridSim 5.2 on a single link

- Packet-level:

- Flow sharing:

CloudSim on a single link

Validity of GridSim and CloudSim

"*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.*"

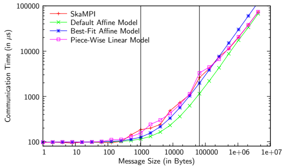
GridSim 5.2 on a single link

- Packet-level: latency paid for every packet, not only the first one
  Expected: $T = lat + \frac{size}{BW}$; Observed: $T = n \times lat + \frac{size}{BW}$
- Flow sharing: buggy, code intend seems ok, but fails on tests
  Expected: $\rho_i = \frac{BW}{n}$; Observed: $\rho_1 = \frac{BW}{1}$; $\rho_2 = \frac{BW}{2}$; $\rho_3 = \frac{BW}{3}$; $\rho_4 = \frac{BW}{4}$; $\ldots$

CloudSim on a single link

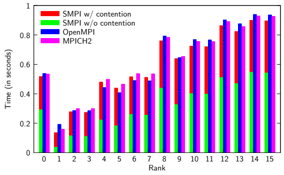
- Sharing unchanged when flows start or end $\Rightarrow$ no sharing between t and $t + \epsilon$
- Most simulated scenarios are then as realistic as a dice roll
  (you could implement you own model in GridSim and CloudSim, but hey)

Accuracy of simulations with SimGrid



Timings of each MPI communication

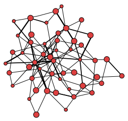
- $\lambda + size \times \tau$ not sufficient (TCP congestion)
- No affine fonction can match for all message sizes
- A 3-parts piecewise affine gives satisfying results

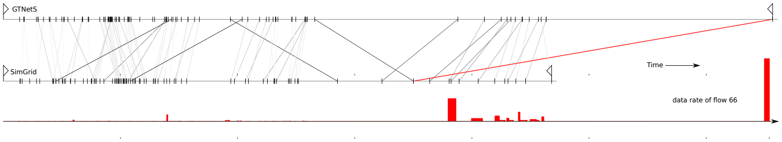


Taking resource sharing into account

- Cannot ignore contention (as other simulators do)
- Our "error" ≈ difference between runtimes
- This is only one collective; full application in progress

Invalidation studies (synthetic invalidating XP for SimGrid)





SimGrid Scalability

Simulation Versatility should not hinder Scalability

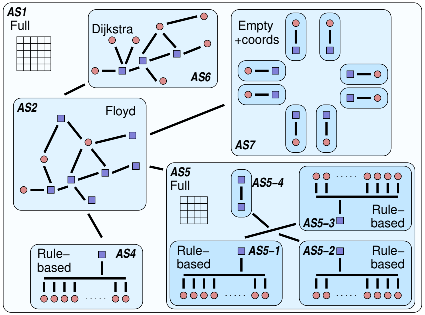
- Two aspects: Big enough (large platforms) ⊕ Fast enough (large workload)

Versatile yet Scalable Platform Descriptions

- Hierarchical organization in ASes
  ⇝ cuts down complexity
  ⇝ recursive routing
- Efficient on each classical structures
  Flat, Floyd, Star, Coordinate-based
- Allow bypass at any level

⇝ Grid'5000 platform in 22KiB
(10 sites, 40 clusters, 1500 nodes)

⇝ King's dataset in 290KiB
(2500 nodes, coordinate-based)



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

Comparison to GridSim

A master distributes 500, 000 fixed size jobs to 2, 000 workers (round robin)

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

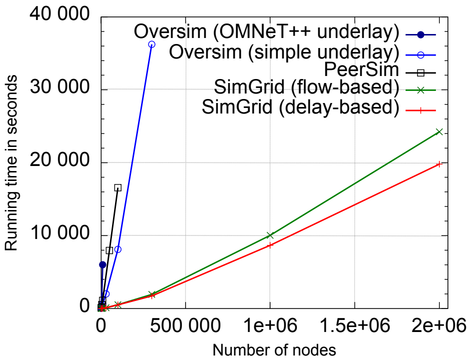
Volunteer Computing settings

- Loosely coupled scenario as in Boinc
- SimGrid: full modeling (clients and servers), precise network model
- SimBA: Servers only, descisions based on simplistic markov modeling

$\leadsto$ SimGrid shown 25 times faster

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

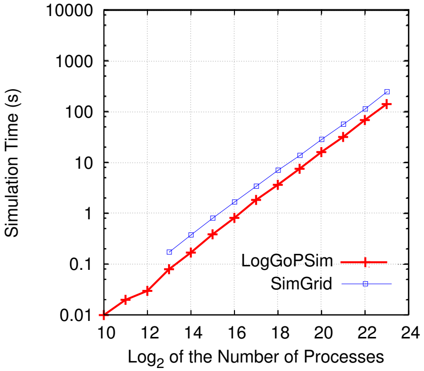
Simulator	size	time
OverSim (OMNeT++)	10k	1h40
OverSim (simple)	300k	10h
PeerSim	100k	4h36
SG (flow-based)	10k	130s
	300k	32mn
	2M*	6h23
SG (delay-based)	2M	5h30

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

- Orders of magnitude more scalable than state-of-the-art P2P simulators
- Precise model incurs a ≈ 20% slowdown, but accuracy is not comparable
- Also, parallel simulation (faster simulation at scale); Distributed sim. ongoing

How big and how fast? (3/3 – HPC)

Simulating a binomial broadcast



Model:

- SimGrid: contention + cabinets hierarchy
- LogGOPSim: simple delay-based model

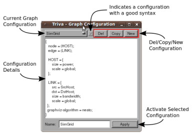
Results:

- SimGrid is roughly 75% slower
- SimGrid is about 20% more fat (15GB required for 2^{23} processors)

The genericity of SimGrid data structures comes at the cost of a slight overhead BUT scalability does not necessarily comes at the price of realism

Visualizing SimGrid Simulations

- Visualization scriptable: easy but powerful configuration; Scalable tools
- Right Information: both platform and applicative visualizations
- Right Representation: gantt charts, spatial representations, tree-graphs
- Easy navigation in space and time: selection, aggregation, animation
- Easy trace comparison: Trace diffing (still partial ATM)



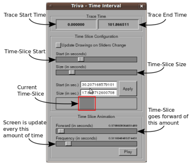
Current Graph Configuration
Indicates a configuration with a good syntax
Del/Copy/New Configuration
Configuration Details
Activate Selected Configuration
Triva - Graph Configuration
SimGrid
Del
Copy
New
Name:
Apply

```
{
  node = (HOST);
  edge = (LINK);

  HOST = {
    size = power;
    scale = global;
  };

  LINK = {
    src = SrcHost;
    dst = DstHost;
    size = bandwidth;
    scale = global;
  };
  graphviz-algorithm = neato;
}
```

Name: SimGrid



Trace Time: 0.000000 — 101.866511

Start (in seconds)

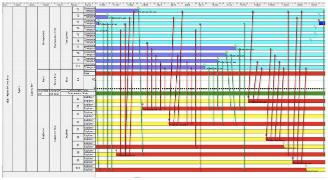
Size (in seconds)

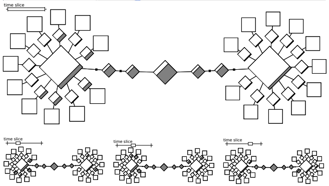
Start (in sec.) 30.207168579101

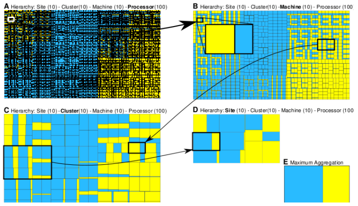
Size (in sec.) 17.8?9712600708

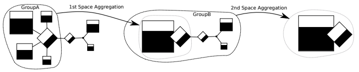
Forward (in seconds) 0.5798608064651489

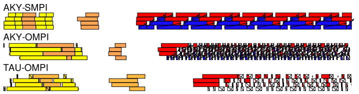
Frequency (in seconds) 0.5116948485374451







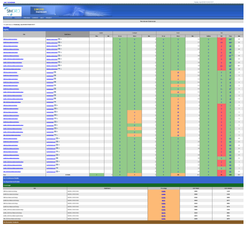




Practical Trust onto SimGrid?

- Internal code base rather complex because of hacks for versatile efficiency

Continuous Integration



- Current version tested every night
- 250 integration tests; 10,000 unit tests; 70% coverage
- 2 SimGrid configurations on 10 Linux versions
- Performance regression testing soon operational

Release tests

- Windows and Mac considered as additional release goals
- Actually works on all Debian arch.: hurd, kfreebsd, mips, arm, ppc, s390 ;)

This is free software anyway

- The code base is currently LGPL (probably soon GPL)
- Come, check it out and participate! (5 of 25 commiters not affiliated to us)

SimGrid goes to the Clouds

EC2-like interface provided for VM manipulation

- vm=VM_start(host, coreAmount)
- VM_bind(vm, process); VM_unbind(vm, process)
- VM_suspend(vm); VM_resume(vm)
- VM_migrate(vm, host)
- VM_shutdown(vm); VM_reboot(vm); VM_destroy(vm)

- This can be used jointly with the Disk storage API for increased realism

$\rightsquigarrow$ Simple, but hopefully sufficient (and lasting) base interface

Work in progress: toward a full Cloud Broker simulation

- Jonathan Rouzaud currently working on Bag-of-Task scheduling on the Cloud
- Algorithms: 7 provisionning, 18 allocation; Models: 3 pricing
- TODO: VM interaction modeling; Synthetic workload for provider-side
- Come and join the discussion!

Take Away Messages

SimGrid will prove helpful to your research

- Versatile: Used in several communities (scheduling, GridRPC, HPC, P2P, Clouds)
- Accurate: Model limits known thanks to validation studies
- Sound: Easy to use, extensible, fast to execute, scalable to death, well tested
- Open: User-community much larger than contributors group; LGPL
- Around since over 10 years, and ready for at least 10 more years

Welcome to the Age of (Sound) Computational Science

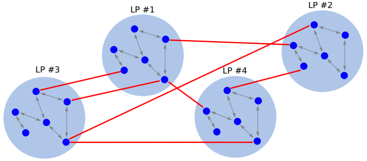
- Discover: `http://simgrid.gforge.inria.fr/`
- Learn: 101 tutorials, user manuals and examples
- Join: user mailing list, #simgrid on irc.debian.org
  We even have some open positions ;)

Question slides

Parallel P2P simulators: the dPeerSim attempt

dPeerSim

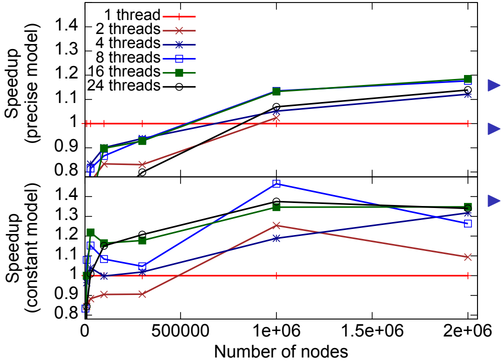
- Parallel implementation of PeerSim/DES (not by PeerSim main authors)
- Classical parallelization: spreads the load over several Logical Processes (LP)



Experimental Results

- Uses Chord as a standard workload: e.g. 320,000 nodes $\rightsquigarrow$ 320,000 requests
- Very good speedup results: 4h10 on 2 LPs, only 1h06 using 16 LPs
- But 47s in the original sequential PeerSim (and 5s in precise SimGrid)
- Yet, best previously known parallelization of DES simulator of P2P systems

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

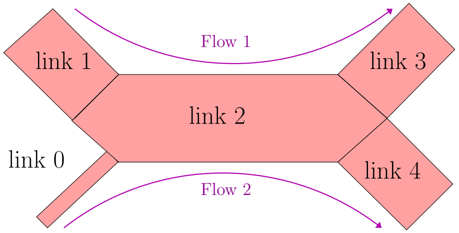
Parallel Efficiency ($\frac{speedup}{\#cores}$) for 2M nodes

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

Max-Min Fairness on Backbone



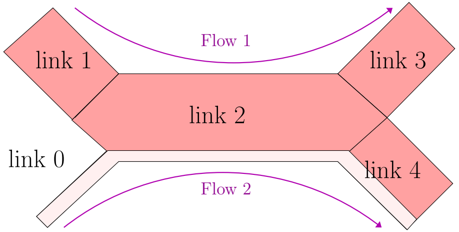
$C_0 = 1$	$n_0 = 1$
$C_1 = 1000$	$n_1 = 1$
$C_2 = 1000$	$n_2 = 2$
$C_3 = 1000$	$n_3 = 1$
$C_4 = 1000$	$n_4 = 1$

$\rho_1 =$

$\rho_2 =$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_2 = 1$. Update the links
- The limiting link is link 2 $\left(\text{since } \frac{999}{1} = min\left(\frac{1000}{1}, \frac{999}{1}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_1 = 999$
- Done. We know ρ_1 and ρ_2

Max-Min Fairness on Backbone



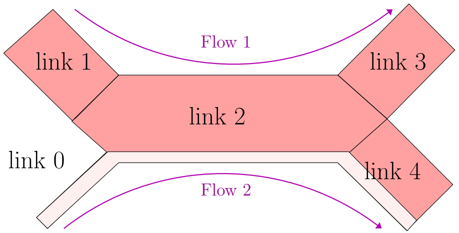
$C_0 = 0$ $n_0 = 0$
$C_1 = 1000$ $n_1 = 1$
$C_2 = 999$ $n_2 = 1$
$C_3 = 1000$ $n_3 = 1$
$C_4 = 999$ $n_4 = 0$

$\rho_1 =$

$\rho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_2 = 1$. Update the links
- The limiting link is link 2 $\left(\text{since } \frac{999}{1} = min\left(\frac{1000}{1}, \frac{999}{1}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_1 = 999$
- Done. We know ρ_1 and ρ_2

Max-Min Fairness on Backbone



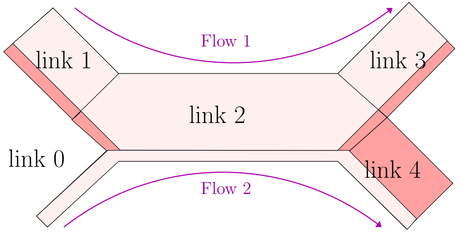
$C_0 = 0$	$n_0 = 0$
$C_1 = 1000$	$n_1 = 1$
$C_2 = 999$	$n_2 = 1$
$C_3 = 1000$	$n_3 = 1$
$C_4 = 999$	$n_4 = 0$

$\rho_1 =$

$\rho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_2 = 1$. Update the links
- The limiting link is link 2 $\left(\text{since } \frac{999}{1} = min\left(\frac{1000}{1}, \frac{999}{1}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_1 = 999$
- Done. We know ρ_1 and ρ_2

Max-Min Fairness on Backbone



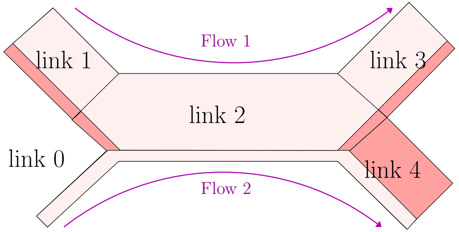
$C_0 = 0$	$n_0 = 0$
$C_1 = 1$	$n_1 = 0$
$C_2 = 0$	$n_2 = 0$
$C_3 = 1$	$n_3 = 0$
$C_4 = 999$	$n_4 = 0$

$\rho_1 = 999$

$\rho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_2 = 1$. Update the links
- The limiting link is link 2 $\left(\text{since } \frac{999}{1} = min\left(\frac{1000}{1}, \frac{999}{1}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_1 = 999$
- Done. We know ρ_1 and ρ_2

Max-Min Fairness on Backbone



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

$\rho_1 = 999$

$\rho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_2 = 1$. Update the links
- The limiting link is link 2 $\left(\text{since } \frac{999}{1} = min\left(\frac{1000}{1}, \frac{999}{1}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\rho_1 = 999$
- Done. We know ρ_1 and ρ_2

Validity of OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

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: $\rho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$

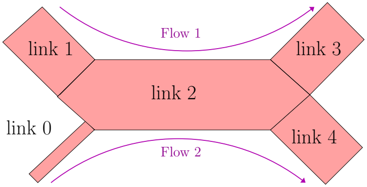
Validity of OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

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: $\rho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



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

$\rho_1 =$

$\rho_2 =$

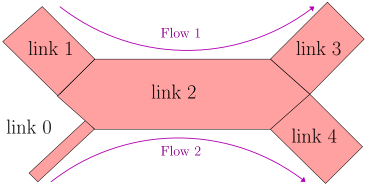
Validity of OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

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: $\rho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$\begin{array}{lll} C_0 = 1 & n_1 = 1 & \text{share} = 1 \\ C_1 = 1000 & n_1 = 1 & \text{share} = 1000 \\ C_2 = 1000 & n_2 = 2 & \text{share} = 500 \\ C_3 = 1000 & n_3 = 1 & \text{share} = 1000 \\ C_4 = 1000 & n_4 = 1 & \text{share} = 1000 \end{array}$$

$$\rho_1 = min(1000, 500, 1000)$$
$$\rho_2 = min(\ 1 \quad , 500, 1000)$$

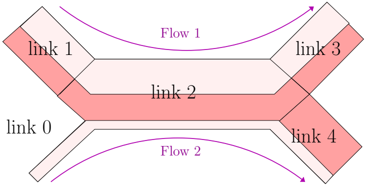
Validity of OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

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: $\rho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$\begin{array}{lll} C_0 = 1 & n_1 = 1 & \text{share} = 1 \\ C_1 = 1000 & n_1 = 1 & \text{share} = 1000 \\ C_2 = 1000 & n_2 = 2 & \text{share} = \mathit{500} \\ C_3 = 1000 & n_3 = 1 & \text{share} = 1000 \\ C_4 = 1000 & n_4 = 1 & \text{share} = 1000 \end{array}$$

$\rho_1 = min(1000, 500, 1000) = \mathbf{500}!!$
$\rho_2 = min(\ 1 \quad, 500, 1000) = 1$

ρ_1 limited by link 2, but 499 still unused on link 2

This "unwanted feature" is even listed in the README file...

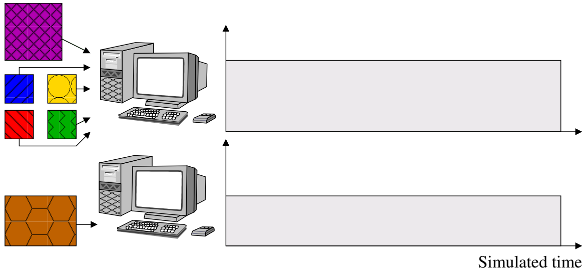
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

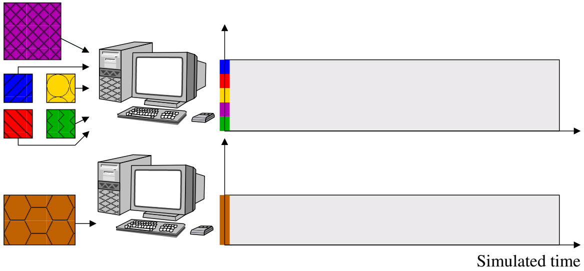
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

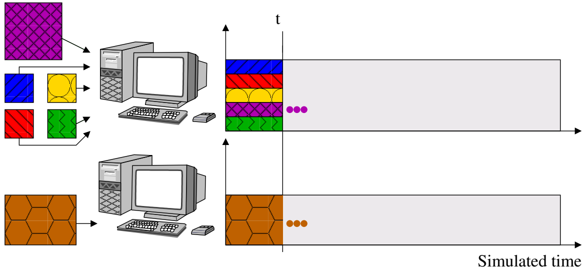
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

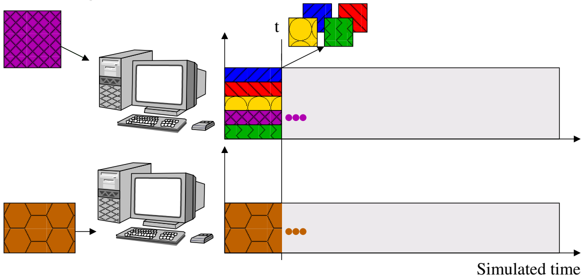
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time $= A/R$

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

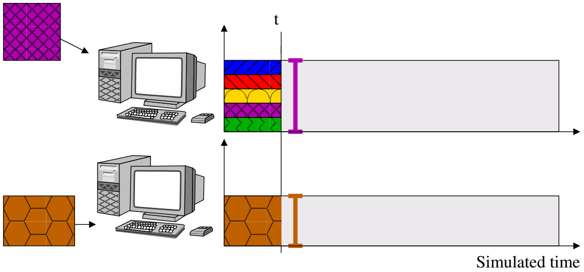
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

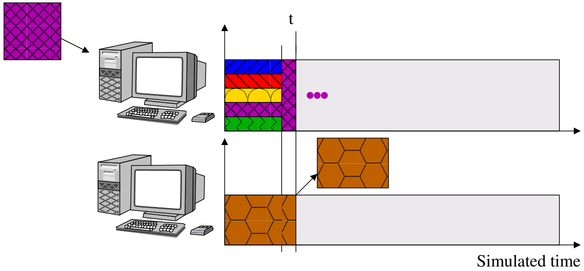
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

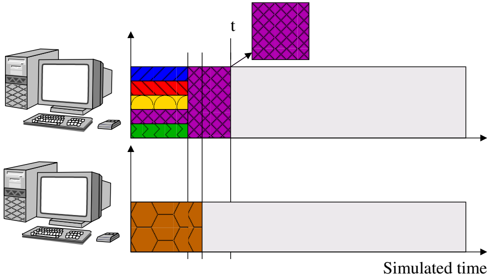
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex

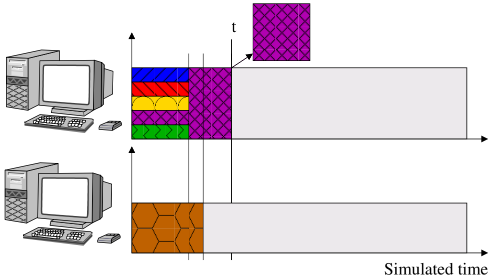
The CPU model in a Nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



In addition in SimGrid

- Availabilities & Failures Traces and Generators
- Sharing for networks is a bit more complex