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,
Cambrige, 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

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000

- Large platforms: getting access is problematic, fixed experimental settings

Some Existing Experimental Tools

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
PlanetLab	virtualize	virtualize	virtualize	virtualize	access	uncontrolled	hundreds

- Large platforms: getting access is problematic, fixed experimental settings
- Virtualization: no control over experimental settings

Some Existing Experimental Tools

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
PlanetLab	virtualize	virtualize	virtualize	virtualize	access	uncontrolled	hundreds
ModelNet	-	-	emulation	emulation	lot material	controlled	dozens
MicroGrid	emulation	-	fine d.e.	emulation	none	controlled	hundreds

- 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

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

- 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

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

- 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

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

- 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

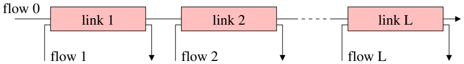
	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
PlanetLab	virtualize	virtualize	virtualize	virtualize	access	uncontrolled	hundreds
ModelNet	-	-	emulation	emulation	lot material	controlled	dozens
MicroGrid	emulation	-	fine d.e.	emulation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

- 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

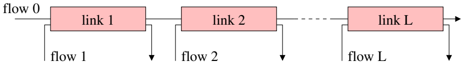
- Data streams modeled as fluids in pipes



Analytical Network Models

Analytical Models proposed in literature

- Data streams modeled as fluids in pipes



Max-Min Fairness

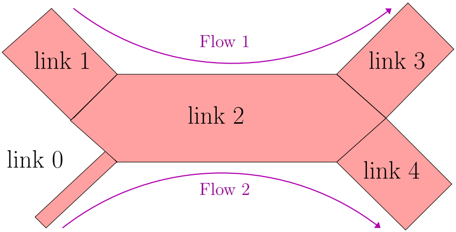
- One of the possible way to compute the transfer rates (λ_f)
- Objective function: maximize $\min_{f \in \mathcal{F}}(\lambda_f)$
- Equilibrium reached if unable to increase any rate without decreasing another
- Gives a fair share to everyone

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; λ_f : transfer rate of f.



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

$\lambda_1 =$
$\lambda_2 =$

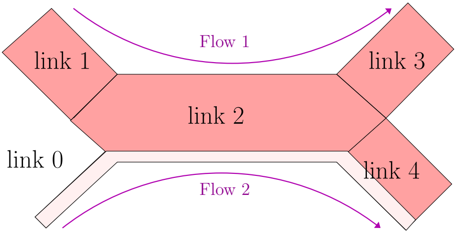
- The limiting link is 0

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; λ_f : transfer rate of f.



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

$\lambda_1 =$
$\lambda_2 = 1$

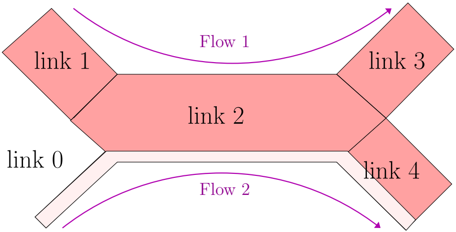
- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links

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; λ_f : transfer rate of f.



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

$\lambda_1 =$
$\lambda_2 = 1$

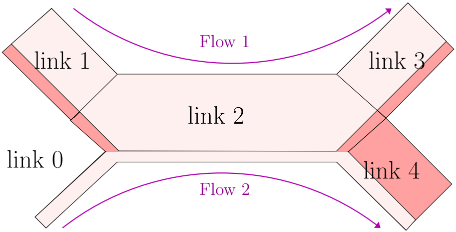
- The limiting link is 0
- This fixes $\lambda_2 = 1$. Update the links
- The limiting link is 2

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_l: capacity of link l; n_l: amount of flows using l; λ_f : transfer rate of f.



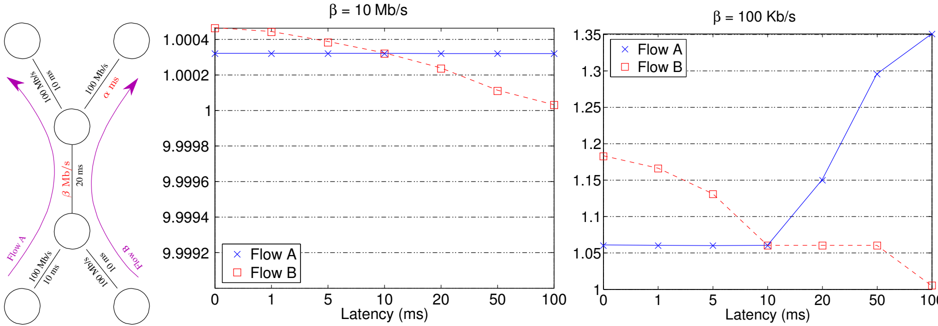
$C_0 = 0$ $n_0 = 0$
$C_1 = 1$ $n_1 = 0$
$C_2 = 0$ $n_2 = 0$
$C_3 = 1$ $n_3 = 0$
$C_4 = 999$ $n_4 = 0$

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

- 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

	GTNetS		SimGrid	
# of flows	Running time	slowdown	Running time	slowdown
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

100Mb flows

	GTNetS		SimGrid	
# of flows	Running time	slowdown	Running time	slowdown
10	65s	0.92	0.001s	0.00002
100	753s	8.08	0.138s	0.00142
200	1562s	12.59	0.538s	0.00402

- 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

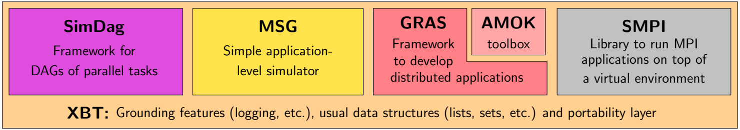
Want more?

- 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

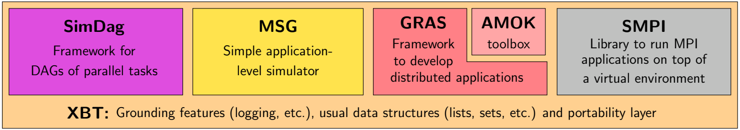
User-visible SimGrid Components



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



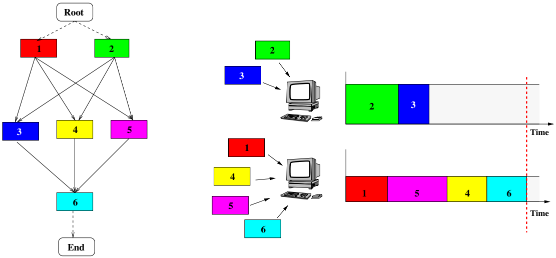
SimGrid user APIs

- 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

SimDag: Comparing Scheduling Heuristics for DAGs



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)
   $\rightsquigarrow$ 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

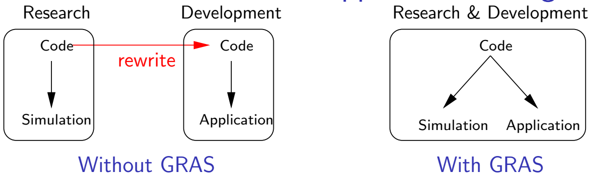
- **Agent:** some code, some private data, running on a given host
- **Task:** amount of work to do and of data to exchange
- Host: location on which agents execute
- Channel: mailbox number on an host (MPI tag)

Usage

- Was used for Grid Scheduling, Desktop Grid, P2P Systems, ...
  (grounded $\approx$ 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

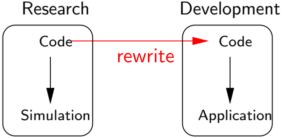


Framework for Rapid Development of Distributed Infrastructure

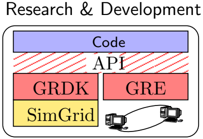
- Develop and tune on the simulator; Deploy *in situ* without modification

GRAS (Grid Reality And Simulation)

Ease development of real distributed applications using a simulator



Without GRAS



With GRAS

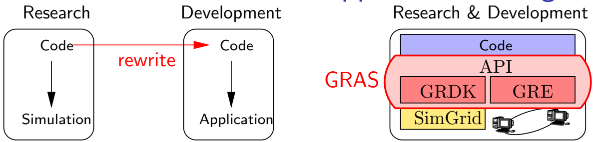
Framework for Rapid Development of Distributed Infrastructure

- 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



Without GRAS   With GRAS

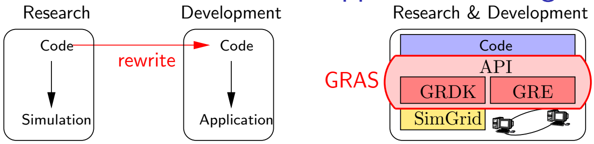
Framework for Rapid Development of Distributed Infrastructure

- 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



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

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

- Performance concern: efficient communication of structured data

  How: Efficient wire protocol (avoid data conversion when possible)
- Portability concern: because of grid heterogeneity

$\leadsto$ 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

# tasks	Native version	Java version
1,000	0.7s	0.5s
10,000	1.7s	2.5s
100,000	9.6s	23s
1,000,000	96s	240s

- 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

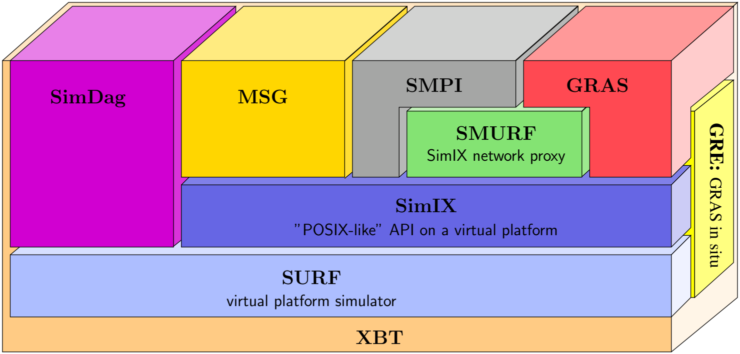
- SimDag: Comparing Scheduling Heuristics for DAGs of (parallel) tasks
- MSG: Comparing Heuristics for Concurrent Sequential Processes
- GRAS: Developing and Debugging Real Applications
- Other ones coming: SMPI, *BSP, OpenMP*

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



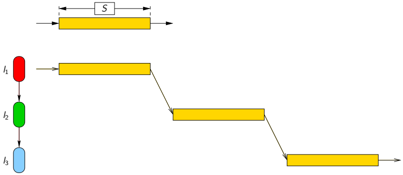
http://simgrid.gforge.inria.fr/

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)

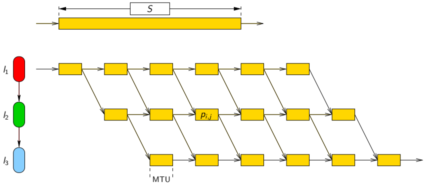


Wormhole Model

(used in GridSim and ChicSim)

- As slow as packet-level
- TCP congestion mechanism neglected

⇒ Poor accuracy



Side note: OptorSim 2.1 on Backbone

OptorSim (developped @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: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$

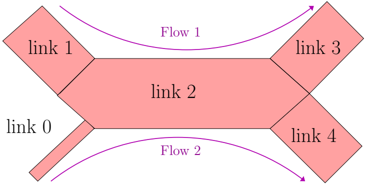
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @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: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$C_0 = 1$ $n_1 = 1$ 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 =

$\lambda_1 =$

$\lambda_2 =$

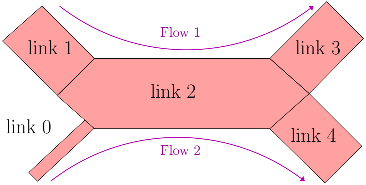
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @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: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$C_0 = 1$	$n_1 = 1$	share $= 1$
$C_1 = 1000$	$n_1 = 1$	share $= 1000$
$C_2 = 1000$	$n_2 = 2$	share $= 500$
$C_3 = 1000$	$n_3 = 1$	share $= 1000$
$C_4 = 1000$	$n_4 = 1$	share $= 1000$

$\lambda_1 = min(1000, 500, 1000)$

$\lambda_2 = min(\ 1\quad, 500, 1000)$

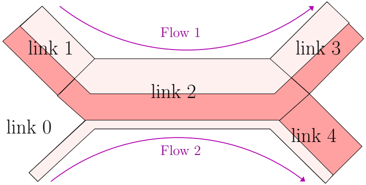
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @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: $\lambda_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$C_0 = 1$	$n_1 = 1$	share $= 1$
$C_1 = 1000$	$n_1 = 1$	share $= 1000$
$C_2 = 1000$	$n_2 = 2$	share $=$ *500*
$C_3 = 1000$	$n_3 = 1$	share $= 1000$
$C_4 = 1000$	$n_4 = 1$	share $= 1000$

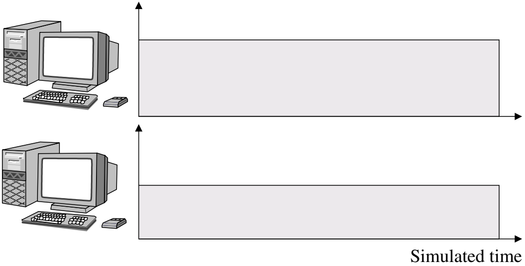
$\lambda_1 = min(1000, 500, 1000) =$ **500**!!

$\lambda_2 = min(\ 1 \quad, 500, 1000) = 1$

Listed as "unwanted feature" in the README file...

Simulation Main Loop

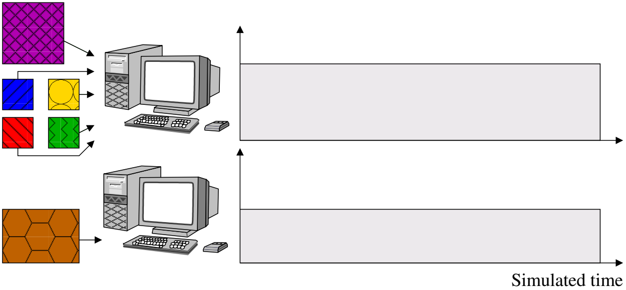
Data: set of resources with working rate



Simulation Main Loop

Data: set of resources with working rate

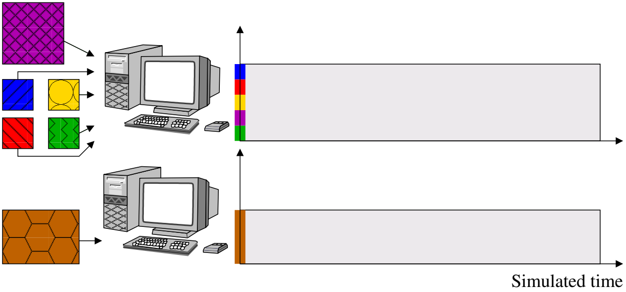
1. Some actions get created and assigned to resources



Simulation Main Loop

Data: set of resources with working rate

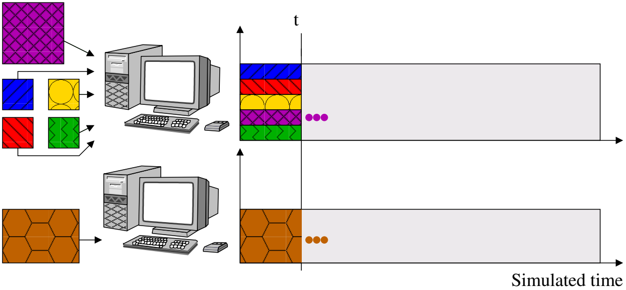
1. Some actions get created and assigned to resources
2. Compute share of everyone (resource sharing algorithms)



Simulation Main Loop

Data: set of resources with working rate

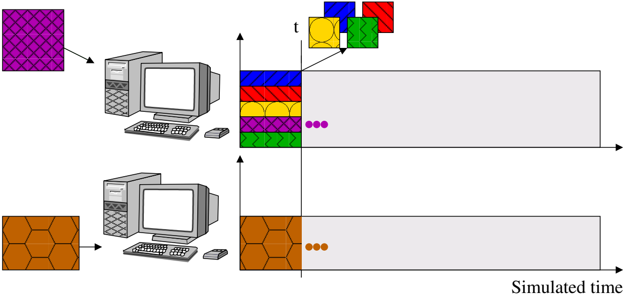
1. Some actions get created and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time



Simulation Main Loop

Data: set of resources with working rate

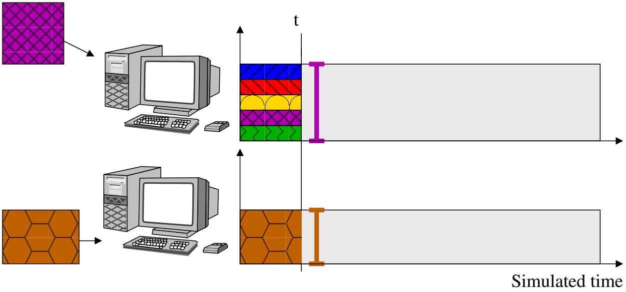
1. Some actions get created and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions



Simulation Main Loop

Data: set of resources with working rate

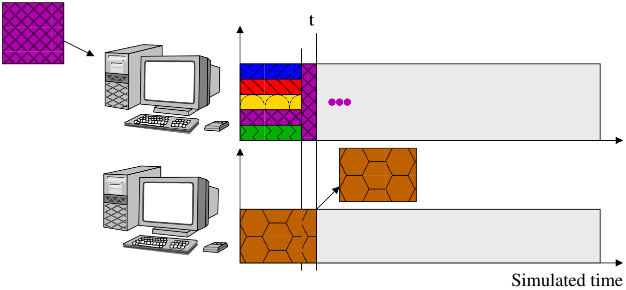
1. Some actions get created and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



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



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