The SimGrid Framework for Research on Large-Scale Distributed Systems

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

simgrid-dev@gforge.inria.fr

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 range 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

Large-Scale Distributed Systems Science?

Requirement for a Scientific Approach

- Reproducible results
  - You can read a paper,
  - reproduce a subset of its results,
  - improve
- Standard methodologies and tools
  - Grad students can learn their use and become operational quickly
  - Experimental scenario can be compared accurately

Current practice in the field: quite different

- Very little common methodologies and tools
- Experimental settings rarely detailed enough in literature (test source codes?)

Purpose of this tutorial

- Present "emerging" methodologies and tools
- Show how to use some of them in practice
- Discuss open questions and future directions

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
    - Real-world experiments
    - Simulation
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Analytical or Experimental?

Analytical works?

- Some purely mathematical models exist
- ☺ Allow better understanding of principles in spite of dubious applicability
  impossibility theorems, parameter influence, ...
- ☹ Theoretical results are difficult to achieve
  - Everyday practical issues (routing, scheduling) become NP-hard problems
    Most of the time, only heuristics whose performance have to be assessed are proposed
  - Models too simplistic, rely on ultimately unrealistic assumptions.

⇒ One must run experiments

⇝ Most published research in the area is experimental

Running real-world experiments

- ☺ Eminently *believable* to demonstrate the proposed approach applicability
- ☹ Very time and labor consuming
  - Entire application must be functional
  - Parameter-sweep; Design alternatives
- ☹ Choosing the right testbed is difficult
  - My own little testbed?
    - ☺ Well-behaved, controlled,stable
    - ☹ Rarely representative of production platforms
  - Real production platforms?
    - Not everyone has access to them; CS experiments are disruptive for users
    - Experimental settings may change drastically during experiment
      (components fail; other users load resources; administrators change config.)
- ☹ Results remain limited to the testbed
  - Impact of testbed specificities hard to quantify ⇒ collection of testbeds...
  - Extrapolations and explorations of "what if" scenarios difficult
    (what if the network were different? what if we had a different workload?)
- ☹ Experiments are uncontrolled and unrepeatable
  No way to test alternatives back-to-back (even if disruption is part of the experiment)

Difficult for others to reproduce results
even if this is the basis for scientific advances!

Simulation

☺ Simulation solves these difficulties

- No need to build a real system, nor the full-fledged application
- Ability to conduct controlled and repeatable experiments
- (Almost) no limits to experimental scenarios
- Possible for anybody to reproduce results

Simulation in a nutshell

- *Predict aspects of the behavior of a system using an approximate model of it*
- Model: Set of objects defined by a state ⊕ Rules governing the state evolution
- Simulator: Program computing the evolution according to the rules
- Wanted features:
  - Accuracy: Correspondence between simulation and real-world
  - Scalability: Actually usable by computers (fast enough)
  - Tractability: Actually usable by human beings (simple enough to understand)
  - Instanciability: Can actually describe real settings (no magical parameter)
  - Relevance: Captures object of interest

Simulation in Computer Science

Microprocessor Design

- A few standard "cycle-accurate" simulators are used extensively
  `http://www.cs.wisc.edu/~arch/www/tools.html`

⇒ Possible to reproduce simulation results

Networking

- A few established "packet-level" simulators: NS-2, DaSSF, OMNeT++, GTNetS
- Well-known datasets for network topologies
- Well-known generators of synthetic topologies
- SSF standard: `http://www.ssfnet.org/`

⇒ Possible to reproduce simulation results

Large-Scale Distributed Systems?

- No established simulator up until a few years ago
- Most people build their own "ad-hoc" solutions

Simulation in Parallel and Distributed Computing

- Used for decades, but under drastic assumptions in most cases

Simplistic platform model

- Fixed computation and communication rates (Flops, Mb/s)
- Topology either fully connected or bus (no interference or simple ones)
- Communication and computation are perfectly overlappable

Simplistic application model

- All computations are CPU intensive (no disk, no memory, no user)
- Clear-cut communication and computation phases
- Computation times even ignored in Distributed Computing community
- Communication times sometimes ignored in HPC community

Straightforward simulation in most cases

- Fill in a Gantt chart or count messages with a computer rather than by hand
- No need for a "simulation standard"

Large-Scale Distributed Systems Simulations?

Simple models justifiable at small scale

- Cluster computing (matrix multiply application on switched dedicated cluster)
- Small scale distributed systems

Hardly justifiable for Large-Scale Distributed Systems

- Heterogeneity of components (hosts, links)
  - Quantitative: CPU clock, link bandwidth and latency
  - Qualitative: ethernet *vs* myrinet *vs* quadrics; Pentium *vs* Cell *vs* GPU
- Dynamicity
  - Quantitative: resource sharing $\rightsquigarrow$ availability variation
  - Qualitative: resource come and go (churn)
- Complexity
  - Hierarchical systems: grids of clusters of multi-processors being multi-cores
  - Resource sharing: network contention, QoS, batches
  - Multi-hop networks, non-negligible latencies
  - Middleware overhead (or optimizations)
  - Interference of computation and communication (and disk, memory, etc)

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
    - Possible designs
    - Experimentation platforms: Grid'5000 and PlanetLab
    - Emulators: ModelNet and MicroGrid
    - Packet-level Simulators: ns-2, SSFNet and GTNetS
    - Ad-hoc simulators: ChicagoSim, OptorSim, GridSim, ...
    - Peer to peer simulators
    - SimGrid
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - Comparing Heuristics for Concurrent Sequential Processes

Models of Large-Scale Distributed Systems

Model = Set of objects defined by a state ⊕ Set of rules governing the state evolution

Model objects:

- Evaluated application: Do actions, stimulus to the platform
- Resources (network, CPU, disk): Constitute the platform, react to stimulus.
  - Application blocked until actions are done
  - Resource can sometime "do actions" to represent external load

Expressing interaction rules

more abstract ↑	Mathematical Simulation: Based solely on equations
	Discrete-Event Simulation: System = set of dependant actions & events
	Emulation: Trapping and virtualization of low-level application/system actions
less abstract ↓	Real execution: No modification

Boundaries are blurred

- Tools can combine several paradigms for different resources
- Emulators may use a simulator to compute resource availabilities

Simulation options to express rules

Network

- Macroscopic: Flows in "pipes" (mathematical & coarse-grain d.e. simulation)
  Data sizes are "liquid amount", links are "pipes"
- Microscopic: Packet-level simulation (fine-grain d.e. simulation)
- Emulation: Actual flows through "some" network timing + time expansion

CPU

- Macroscopic: Flows of operations in the CPU pipelines
- Microscopic: Cycle-accurate simulation (fine-grain d.e. simulation)
- Emulation: Virtualization via another CPU / Virtual Machine

Applications

- Macroscopic: Application = analytical "flow"
- Less macroscopic: Set of abstract tasks with resource needs and dependencies
  - Coarse-grain d.e. simulation
  - Application specification or pseudo-code API
- Virtualization: Emulation of actual code trapping application generated events

Large-Scale Distributed Systems Simulation Tools

A lot of tools exist

- Grid'5000, Planetlab, MicroGrid, Modelnet, Emulab, DummyNet
- ns-2, GTNetS, SSFNet
- ChicagoSim, GridSim, OptorSim, SimGrid, ...
- PeerSim, P2PSim, ...

How do they compare?

- How do they work?
  - Components taken into account (CPU, network, application)
  - Options used for each component (direct execution; emulation; d.e.; simulation)
- What are their relative qualities?
  - Accuracy (correspondence between simulation and real-world)
  - Technical requirement (programming language, specific hardware)
  - Scale (tractable size of systems at reasonable speed)
  - Experimental settings configurable and repeatable, or not

Experimental tools comparison

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
Planetlab	virtualize	virtualize	virtualize	virtualize	none	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++ and 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
ChicSim	coarse d.e.	-	coarse d.e.	coarse d.e.	C	controlled	few 1,000
OptorSim	coarse d.e.	amount	coarse d.e.	coarse d.e.	Java	controlled	few 1,000
GridSim	coarse d.e.	coarse d.e.	coarse d.e.	coarse d.e.	Java	controlled	few 1,000
P2PSim	-	-	-	state machine	C++	controlled	few 1,000
PlanetSim	-	-	cste time	coarse d.e.	Java	controlled	100,000
PeerSim	-	-	-	state machine	Java	controlled	1,000,000
SimGrid	math/d.e.	(underway)	math/d.e.	d.e./emul	C or Java	controlled	few 100,000

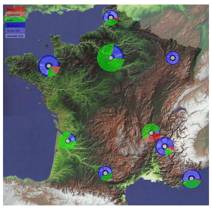
- Direct execution $\rightsquigarrow$ no experimental bias (?)
  Experimental settings fixed (between hardware upgrades), but not controllable
- Virtualization allows sandboxing, but no experimental settings control
- Emulation can have high overheads (but captures the overhead)
- Discrete event simulation is slow, but hopefully accurate
  To scale, you have to trade speed for accuracy

Grid'5000 (consortium – INRIA)

French experimental platform

- 1500 nodes (3000 cpus, 4000 cores) over 9 sites
- Nation-wide 10Gb dedicated interconnection
- `http://www.grid5000.org`

Scientific tool for computer scientists

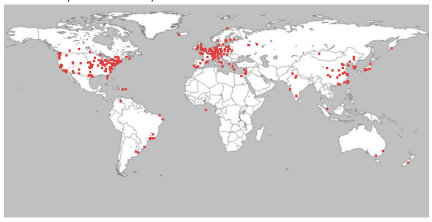


- Nodes are *deployable*: install your own OS image
- Allow study at any level of the stack:
  - Network (TCP improvements)
  - Middleware (scalability, scheduling, fault-tolerance)
  - Programming (components, code coupling, GridRPC)
  - Applications
- ☺ Applications not modified, direct execution
- ☺ Environment controlled, experiments repeatable
- ☹ Relative scalability ("only" 1500-4000 nodes)

PlanetLab (consortium)

Open platform for developping, deploying, and accessing planetary-scale services

Planetary-scale 852 nodes, 434 sites, >20 countries



Distribution Virtualization each user can get a slice of the platform

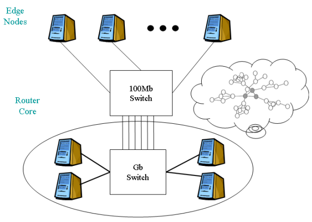
Unbundled Management
- local behavior defined per node; network-wide behavior: services
- multiple competing services in parallel (shared, unprivileged interfaces)

As unstable as the real world
- ☺ Demonstrate the feasability of P2P applications or middlewares
- ☹ No reproducibility!

ModelNet (UCSD/Duke)

Applications

- Emulation and virtualization: Actual code executed on "virtualized" resources
- Key tradeoff: scalability versus accuracy



Resources: system calls intercepted
- gethostname, sockets

CPU: direct execution on CPU
- Slowdown not taken into account!

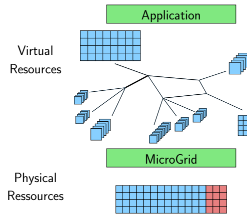
Network: emulation through:
- one emulator (running on FreeBSD)
- a gigabit LAN
- hosts + IP aliasing for virtual nodes
- ⇝ emulation of heterogeneous links
- Similar ideas used in other projects (Emulab, DummyNet, Panda, ...)

Amin Vahdat *et Al.*, *Scalability and Accuracy in a LargeScale Network Emulator*, OSDI'02.

MicroGrid (UCSD)

Applications

- Application supported by emulation and virtualization
- Actual application code is executed on "virtualized" resources
- Accounts for CPU and network



Resources: wraps syscalls & grid tools
- gethostname, sockets, GIS, MDS, NWS

CPU: direct execution on fraction of CPU
- finds right mapping

Network: packet-level simulation
- parallel version of MaSSF

Time: synchronize real and virtual time
- find the good execution rate

Andrew Chien *et Al.*, *The MicroGrid: a Scientific Tool for Modeling Computational Grids*, SuperComputing 2002.

Packet-level simulators

ns-2: the most popular one

- Several protocols (TCP, UDP, ...), several queuing models (DropTail, RED, ...)
- Several application models (HTTP, FTP), wired and wireless networks
- Written in C++, configured using TCL. Limitated scalability ($< 1,000$)

SSFNet: implementation of SSF standard

- Scalable Simulation Framework: unified API for d.e. of distributed systems
- Written in Java, usable on 100 000 nodes

GTNetS: Georgia Tech Network Simulator

- Design close to real networks protocol philosophy (layers stacked)
- C++, reported usable with $177,000$ nodes

Simulation tools of / for the networking community

- Topic: Study networks behavior, routing protocols, QoS, ...
- Goal: Improve network protocols
- ⇝ Microscopic simulation of packet movements
- ⇒ Inadequate for us (long simulation time, CPU not taken into account)

ChicagoSim, OptorSim, GridSim, ...

- Network simulator are not adapted, emulation solutions are too heavy
- PhD students just need simulator to plug in their algorithm
  - Data placement/replication
  - Grid economy
- ⇒ Many simulators. Most are home-made, short-lived; Some are released

ChicSim designed for the study of data replication (Data Grids), built on ParSec
Ranganathan, Foster, *Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications*, HPDC'02.

OptorSim developped for European Data-Grid
DataGrid, CERN. *OptorSim: Simulating data access optimization algorithms*

GridSim focused on Grid economy
Buyya et Al. *GridSim: A Toolkit for the Modeling and Simulation of Global Grids*, CCPE'02.

every [sub-]community seems to have its own simulator

PeerSim, P2PSim, ...

Thee peer-to-peer community also has its own private collection of simulators:

focused on P2P protocols ⇝ main challenge = scale

P2PSim Multi-threaded discrete-event simulator. Constant communication time. Alpha release (april 2005)
`http://pdos.csail.mit.edu/p2psim/`

PlanetSim Multi-threaded discrete-event simulator. Constant communication time. Last release (2006)
`http://planet.urv.es/trac/planetsim/wiki/PlanetSim`

PeerSim Designed for epidemic protocols. processes = state machines. Two simulation modes: cycle-based (time is discrete) or event-based. Resources are not modeled. 1.0.3 release (december 2007)
`http://peersim.sourceforge.net/`

OverSim A recent one based on OMNeT++ (april 2008)
`http://www.oversim.org/`

SimGrid (Hawai'i, Grenoble, Nancy)

History

- Created just like other home-made simulators (only a bit earlier ;)
- Original goal: scheduling research ⇝ need for speed (parameter sweep)
- HPC community concerned by performance ⇝ accuracy not negligible

SimGrid in a Nutshell

- Simulation ≡ communicating processes performing computations
- Key feature: Blend of mathematical simulation and coarse-grain d. e. simulation
- Resources: Defined by a rate (MFlop/s or Mb/s) + latency
  - Also allows dynamic traces and failures
- Tasks can use multiple resources explicitely or implicitly
  - Transfer over multiple links, computation using disk and CPU
- Simple API to *specify* an heuristic or application easily

Casanova, Legrand, Quinson.
SimGrid: a Generic Framework for Large-Scale Distributed Experimentations, EUROSIM'08.

Experimental tools comparison

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000	direct	direct	direct	direct	access	fixed	<5000
Planetlab	virtualize	virtualize	virtualize	virtualize	none	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++ and 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
ChicSim	coarse d.e.	-	coarse d.e.	coarse d.e.	C	controlled	few 1,000
OptorSim	coarse d.e.	amount	coarse d.e.	coarse d.e.	Java	controlled	few 1,000
GridSim	coarse d.e.	coarse d.e.	coarse d.e.	coarse d.e.	Java	controlled	few 1,000
P2PSim	-	-	-	state machine	C++	controlled	few 1,000
PlanetSim	-	-	cste time	coarse d.e.	Java	controlled	100,000
PeerSim	-	-	-	state machine	Java	controlled	1,000,000
SimGrid	math/d.e.	(underway)	math/d.e.	d.e./emul	C or Java	controlled	few 100,000

- Direct execution $\rightsquigarrow$ no experimental bias (?)
  Experimental settings fixed (between hardware upgrades), but not controllable
- Virtualization allows sandboxing, but no experimental settings control
- Emulation can have high overheads (but captures the overhead)
- Discrete event simulation is slow, but hopefully accurate
  To scale, you have to trade speed for accuracy

So what simulator should I use?

It really depends on your goal / resources

- Grid'5000 experiments very good . . . if have access and plenty of time
- PlanetLab does not enable reproducible experiments
- ModelNet, ns-2, SSFNet, GTNetS meant for networking experiments (no CPU)
- ModelNet requires some specific hardware setup
- MicroGrid simulations take a lot of time (although they can be parallelized)
- SimGrid's models have clear limitations (e.g. for short transfers)
- SimGrid simulations are quite easy to set up (but rewrite needed)
- SimGrid does not require that a full application be written
- Ad-hoc simulators are easy to setup, but their validity is still to be shown, *ie,* the results obtained *may* be plainly wrong
- Ad-hoc simulators obviously not generic (difficult to adapt to your own need)

Key trade-off seem to be accuracy *vs* speed

- The more abstract the simulation the fastest
- The less abstract the simulation the most accurate

Does this trade-off really hold?

Simulation Validation

Crux of simulation works

- Validation is difficult
- Almost never done convincingly
- (not specific to CS: other science have same issue here)

How to validate a model (and obtain scientific results?)

- Claim that it is plausible (justification = argumentation)
- Show that it is reasonable
  - Some validation graphs in a few special cases at best
  - Validation against another "validated" simulator
- Argue that trends are respected (absolute values may be off)
  $\rightsquigarrow$ it is useful to compare algorithms/designs
- Conduct extensive verification campaign against real-world settings

Simulation Validation: the FLASH example

FLASH project at Stanford

- Building large-scale shared-memory multiprocessors
- Went from conception, to design, to actual hardware (32-node)
- Used simulation heavily over 6 years

Authors compared simulation(s) to the real world

- Error is unavoidable (30% error in their case was not rare)
  Negating the impact of "we got 1.5% improvement"
- Complex simulators not ensuring better simulation results
  - Simple simulators worked better than sophisticated ones (which were unstable)
  - Simple simulators predicted trends as well as slower, sophisticated ones
  - $\Rightarrow$ Should focus on simulating the important things
- Calibrating simulators on real-world settings is mandatory

For FLASH, the simple simulator was all that was needed. . .

Gibson, Kunz, Ofelt, Heinrich, *FLASH vs. (Simulated) FLASH: Closing the Simulation Loop*, Architectural Support for Programming Languages and Operating Systems, 2000

Conclusion

Large-Scale Distributed System Research is Experimental

- Analytical models are too limited
- Real-world experiments are hard & limited
- $\Rightarrow$ Most literature rely on simulation

Simulation for distributed applications still taking baby steps

- Compared for example to hardware design or networking communities but more advanced for HPC Grids than for P2P
- Lot of home-made tools, no standard methodology
- Very few simulation projects even try to:
  - Publish their tools for others to use
  - Validate their tools
  - Support other people's use:
    genericity, stability, portability, documentation, . . .

Conclusion

Claim: SimGrid may prove helpful to your research

- User-community much larger than contributors group
- Used in several communities (scheduling, GridRPC, HPC infrastructure, P2P)
- Model limits known thanks to validation studies
- Easy to use, extensible, fast to execute
- Around since almost 10 years

Remainder of this talk: present SimGrid in detail

- Under the cover:
  - Models used
  - Implementation overview
- Main limitations
  - Model validity
  - Tool performance and scalability
- Practical usage
  - How to use it for your research
  - Use cases and success stories

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
    - Modeling a Single Resource
    - Multi-hop Networks
    - Resource Sharing
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Analytic Models underlying the SimGrid Framework

Main challenges for SimGrid design

- Simulation accuracy:
  - Designed for HPC scheduling community $\rightsquigarrow$ don't mess with the makespan!
  - At the very least, understand validity range
- Simulation speed:
  - Users conduct large parameter-sweep experiments over alternatives

Microscopic simulator design

- Simulate the packet movements and routers algorithms
- Simulate the CPU actions (or micro-benchmark classical basic operations)
- Hopefully very accurate, but *very* slow (simulation time $\gg$ simulated time)

Going faster while remaining reasonable?

- Need to come up with macroscopic models for each kind of resource
- Main issue: resource sharing. *Emerge* naturally in microscopic approach:
  - Packets of different connections interleaved by routers
  - CPU cycles of different processes get slices of the CPU

Modeling a Single Resource

Basic model: $Time = L + \frac{size}{B}$

- Resource work at given *rate* (B, in MFlop/s or Mb/s)
- Each use have a given latency (L, in s)

Application to processing elements (CPU/cores)

- Very widely used (latency usually neglected)
- No cache effects and other specific software/hardware adequation
- No better analytical model (reality too complex and changing)
- Sharing easy in steady-state: fair share for each process

Application to networks

- Turns out to be "inaccurate" for TCP
- B not constant, but depends on RTT, packet loss ratio, window size, *etc.*
- Several models were proposed in the literature

Modeling TCP performance (single flow, single link)

Padhye, Firoiu, Towsley, Krusoe. *Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation.* IEEE/ACM Transactions on Networking, Vol. 8, Num. 2, 2000.

$$B = min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \times min(1, 3\sqrt{3bp/8}) \times p(1+32p^2)}\right)$$

- W_{max}: receiver advertised window
- p: loss indication rate
- RTT: Round trip time
- b: #packages acknowledged per ACK
- T_0: TCP average retransmission timeout value

Model discussion

- Captures TCP congestion control (fast retransmit and timeout mecanisms)
- Assumes steady-state (no slow-start)
- Accuracy shown to be good over a wide range of values
- p and b not known in general (model hard to instanciable)

SimGrid model for single TCP flow, single link

Definition of the link l

- L_l: physical latency
- B_l: physical bandwidth

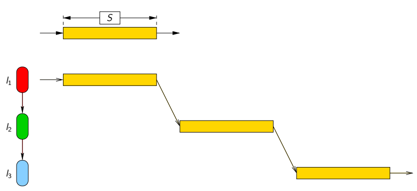
Time to transfer *size* bytes over the link:

$$Time = L_l + \frac{size}{B'_l}$$

Empirical bandwidth: $B'_l = min(B_l, \frac{W_{max}}{RTT})$

- Justification: sender emits W_{max} then waits for ack (ie, waits RTT)
- Upper limit: first *min* member of previous model
- RTT assumed to be twice the physical latency
- Router queue time assumed to be included in this value

Modeling Multi-hop Networks: Store & Forward



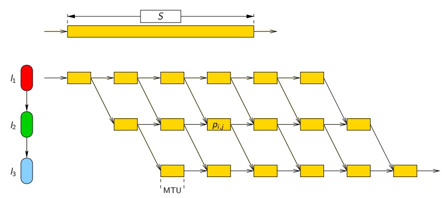
First idea, quite natural

- Pay the price of going through link 1, then go through link 2, etc.
- Analogy to the time to go from a city to another: time on each road

Unfortunately, things don't work this way

- Whole message not stored on each router
- Data split in packets over TCP networks (surprise, surprise)
- Transfers on each link occur in parallel

Modeling Multi-hop Networks: WormHole



Remember Networking classes?

- Links packetize stream according to MTU (Maximum Transmission Unit)
- Easy to simulate (SimGrid until 2002; GridSim 4.0 & most ad-hoc tools do)

Unfortunately, things don't work this way

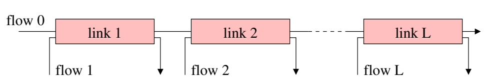
- IP packet fragmentation algorithms complex (when MTUs differ)
- TCP contention mecanisms:
  - Sender only emits *cwnd* packets before ACK
  - Timeouts, fast retransmit, etc.

$\Rightarrow$ as slow as packet-level simulators, not quite as accurate

Macroscopic TCP modeling is a field

TCP bandwidth sharing studied by several authors

- Data streams modeled as fluids in pipes
- Same model for single stream/multiple links or multiple stream/multiple links



Notations

- $\mathcal{L}$: set of links
- $\mathcal{F}$: set of flows; $f \in P(\mathcal{L})$
- C_l: capacity of link l ($C_l > 0$)
- λ_f : transfer rate of f
- n_l: amount of flows using link l

Feasibility constraint

- Links deliver their capacity at most: $\forall l \in \mathcal{L}, \sum_{f \ni l} \lambda_f \leq C_l$

Max-Min Fairness

Objective function: maximize $\min_{f \in \mathcal{F}}(\lambda_f)$

- Equilibrium reached if increasing any λ_f decreases a λ'_f (with $\lambda_f > \lambda'_f$)
- Very reasonable goal: gives fair share to anyone
- Optionally, one can add prorities w_i for each flow i
  $\rightsquigarrow$ maximizing $\min_{f \in \mathcal{F}}(w_f \lambda_f)$

Bottleneck links

- For each flow f, one of the links is the limiting one l
  (with more on that link l, the flow f would get more overall)
- The objective function gives that l is saturated, and f gets the biggest share

$$\forall f \in \mathcal{F}, \exists l \in f, \quad \sum_{f' \ni l} \lambda_{f'} = C_l \quad \text{and} \quad \lambda_f = \max\{\lambda_{f'}, f' \ni l\}$$

L. Massoulié and J. Roberts, *Bandwidth sharing: objectives and algorithms*, IEEE/ACM Trans. Netw., vol. 10, no. 3, pp. 320-328, 2002.

Implementation of Max-Min Fairness

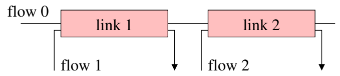
Bucket-filling algorithm

- Set the bandwidth of all flows to 0
- Increase the bandwidth of every flow by ϵ. And again, and again, and again.
- When one link is saturated, all flows using it are limited ($\rightsquigarrow$ removed from set)
- Loop until all flows have found a limiting link

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. $\forall f \in l,\ \ \lambda_f = \frac{C_l}{n_l}$;
   Update all n_l and C_l to remove these flows
3. Loop until all λ_f are fixed

Max-Min Fairness on Homogeneous Linear Network



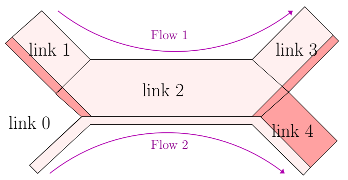
$C_1 = C \quad n_1 = 2$
$C_2 = C \quad n_2 = 2$

$\lambda_0 \leftarrow C/2$
$\lambda_1 \leftarrow C/2$
$\lambda_2 \leftarrow C/2$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1
- $\Rightarrow$ $\lambda_0 = C/2$ and $\lambda_1 = C/2$
- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\lambda_1 = C/2$

We're done computing the bandwidth allocated to each flow

Max-Min Fairness on Backbone



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

$\lambda_1 \leftarrow 999$
$\lambda_2 \leftarrow 1$

- The limiting link is link 0 (since $\frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)$)
- This fixes $\lambda_2 = 1$. Update the links
- The limiting link is link 2
- This fixes $\lambda_1 = 999$
- Done. We know λ_1 and λ_2

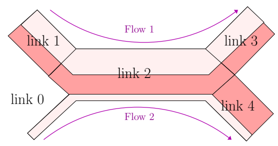
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc 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**!!
$\lambda_2 = min(\ 1\ \ , 500, 1000) = 1$

λ_1 limited by link 2, but 499 still unused on link 2

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

Proportional Fairness

Max-Min validity limits

- MaxMin gives a fair share to everyone
- Reasonable, but TCP does not do so
- Congestion mecanism: Additive Increase, Muplicative Decrease (AIMD)
- Complicates modeling, as shown in literature

Proportional Fairness

- MaxMin gives more to long flows (resource-eager), TCP known to do opposite
- Objective function: maximize $\sum_{\mathcal{F}} w_f \log(\lambda_f)$ (instead of $\min_{\mathcal{F}} w_f \lambda_f$ for MaxMin)
- *log* favors short flows

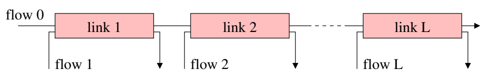
Kelly, *Charging and rate control for elastic traffic*, in European Transactions on Telecommunications, vol. 8, 1997, pp. 33-37.

Implementing Proportional Fairness

Karush Kuhn Tucker conditions:

- Solution $\{\lambda_f\}_{f\in\mathcal{F}}$ is uniq
- Any other feasible solution $\{\lambda'_f\}_{f\in\mathcal{F}}$ satisfy: $\sum_{f\in\mathcal{F}} \frac{\lambda'_f - \lambda_f}{\lambda_f} \leq 0$
- $\Rightarrow$ Compute the point $\{\lambda_f\}$ where the derivate is zero (convex optimization)
- $\rightarrow$ Use Lagrange multipliers and steepest gradient descent

Proportional Fairness on Homogeneous Linear Network



- Maths give that: $\lambda_0 = \frac{C}{n+1}$ and $\forall l \neq 0,\ \lambda_l = \frac{C \times n}{n+1}$
- Ie, for C=100Mb/s and n=3, $\lambda_0 = 25Mb/s$, $\lambda_1 = \lambda_2 = \lambda_3 = 75Mb/s$
- Closer to practitioner expectations

Recent TCP implementation

More protocol refinement, more model complexity

- Every agent changes its window size according to its neighbors' one (selfish net-utility maximization)
- Computing a distributed gradient for Lagrange multipliers $\rightsquigarrow$ same updates

TCP Vegas converges to a weighted proportional fairness

- Objective function: maximize $\sum L_f \times \log(\lambda_f)$ (L_f being the latency)

TCP Reno is even worse

- Objective function: maximize $\sum_{f\in\mathcal{F}} \arctan(\lambda_f)$

Low, S.H., *A Duality Model of TCP and Queue Management Algorithms*, IEEE/ACM Transactions on Networking, 2003.

Efficient implementation: possible, but not so trivial

- Computing *distributed* gradient for Lagrange multipliers: useless in our setting
- Lagrange multipliers computable with efficient optimal-step gradient descent

So, what is the model used in SimGrid?

"`--cfg=network_model`" command line argument

- `CM02` $\rightsquigarrow$ MaxMin fairness
- `Vegas` $\rightsquigarrow$ Vegas TCP fairness (Lagrange approach)
- `Reno` $\rightsquigarrow$ 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` $\rightsquigarrow$ model specific to parallel tasks

Want more?

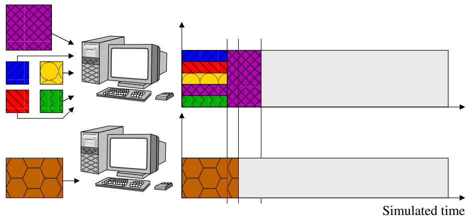
- `network_model:gtnets` $\rightsquigarrow$ use Georgia Tech Network Simulator for network
  Accuracy of a packet-level network simulator without changing your code (!)
- Plug your own model in SimGrid!!
  (usable as scientific instrument in TCP modeling field, too)

How are these models used in practice?

Simulation kernel main loop

Data: set of resources with working rate

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



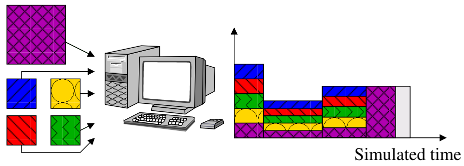
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



SimGrid also accept state changes (on/off)

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
    - Single link
    - Dumbbell
    - Random platforms
    - Simulation speed
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

SimGrid Validation

Quantitative comparison of SimGrid with Packet-Level Simulators

- NS2: The Network Simulator
- SSFnet: Scalable Simulation Framework 2.0 (Dartmouth)
- GTNetS: Georgia Tech Network Simulator

Methodological limits

- Packet-level *supposed* accurate (comparison to real-world: future work)
- Max-Min only: other models were not part of SimGrid at that time

Challenges

- Which topology?
- Which parameters consider? *e.g.* bandwidth, latency, size, all
- How to estimate performance? *e.g.* throughput, communication time
- How to estimate simulation response time slowdown?
- How to compute error? *e.g.* $\sum \frac{Perf_{PacketLevel}}{Perf_{SimGrid}}$

Velho, Legrand, *Accuracy Study and Improvement of Network Simulation in the SimGrid Framework*, to appear in Second International Conference on Simulation Tools and Techniques, SIMUTools'09, Rome, Italy, March 2009.

(other publication by Velho and Legrand submitted to SimuTools'09)

SimGrid Validation

Experiments assumptions

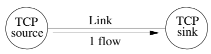
- Topology: Single Link; Dumbbell; Random topologies (several)
- Parameters: data size, #flows, #nodes, link bandwidth and latency
- Performance: communication time and bandwidth estimation
  - All TCP flows start at the same time
  - All TCP flows are stopped when the first flow completes
  - Bandwidth estimation is done based on communication remaining.
- Slowdown: $\frac{\text{Simulation time}}{\text{Simulated time}}$

Notations

- B, link nominal bandwidth ; L, link latency
- S, Amount of transmitted data
- Error: $\varepsilon(T_{GTNetS}, T_{SimGrid}) = \log(T_{GTNetS}) - \log(T_{SimGrid})$
  - Symmetrical for over and under estimations (thanks to logs)
  - Average error: $|\overline{\varepsilon}| = \frac{1}{n}\sum_i |\varepsilon_i|$ Max error: $|\varepsilon_{max}| = \max_i(|\varepsilon_i|)$
  - Computing gain/loss in percentage: $e^{|\overline{\varepsilon}|} - 1$ or $e^{|\varepsilon_{max}|} - 1$

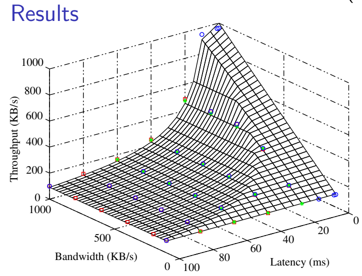
Validation experiments on a single link (1/2)

Experimental settings



- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

Results



Legend

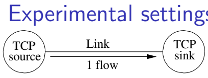
- Mesh: SimGrid results $\frac{S}{S/min(B, \frac{W}{2L}) + L}$
- □: GTNetS results
- ○: NS2 results
- ×: SSFNet with TCP_FAST_INTERVAL=default
- +: SSFNet with TCP_FAST_INTERVAL=0.01

Conclusion

- SimGrid estimations close to packet-level simulators (when S=100MB)
  - When $B < \frac{W}{2L}$ (B=100KB/s, L=500ms), $|\varepsilon_{max}| \approx |\overline{\varepsilon}| \approx 1\%$
  - When $B > \frac{W}{2L}$ (B=100KB/s, L= 10ms), $|\varepsilon_{max}| \approx |\overline{\varepsilon}| \approx 2\%$

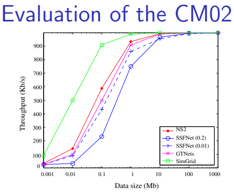
Validation experiments on a single link (2/3)

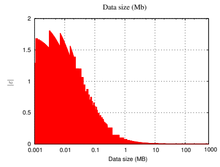
Experimental settings



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

Evaluation of the CM02 model





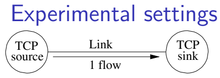
- Packet-level tools don't completely agree
- SSFNet TCP_FAST_INTERVAL bad default
- GTNetS is equally distant from others

- CM02 doesn't take slow start into account

S	$\lvert\overline{\varepsilon}\rvert$	$\lvert\varepsilon_{max}\rvert$
$S < 100KB$	$\approx 146\%$	$\approx 508\%$
$S \in [100KB; 10MB]$	$\approx 17\%$	$\approx 80\%$
$S > 10MB$	$\approx 1\%$	$\approx 1\%$

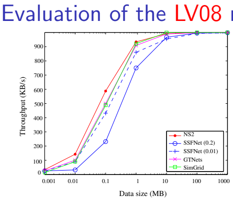
Validation experiments on a single link (3/3)

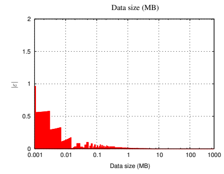
Experimental settings



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

Evaluation of the LV08 model



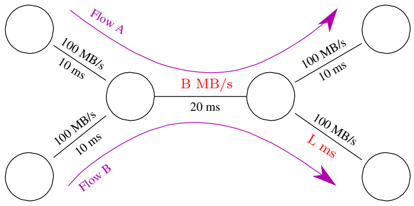


- Statistical analysis of GTNetS slow-start
- New SimGrid model (MaxMin based)
  - Bandwidth decreased (92%)
  - Latency changed to $10.4 \times L$
- This dramatically improve validity range

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

Validation experiments on the dumbbell topology

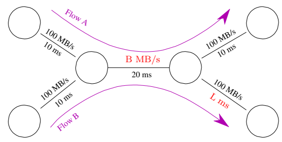
Experimental settings

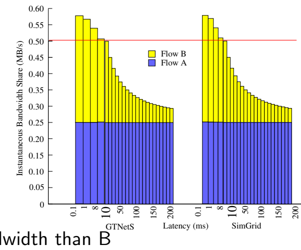


- Comparison limited to the GTNetS packet-level simulator
- Bandwidth: linearly sampled with 16 points B ∈ [0.01, 1000] MB/s
- Latency: linearly sampled with 20 points L ∈ [0, 200] ms
- Size: S=100MB
- Compare instantaneous bandwidth share (SimGrid vs. GTNetS)

Validation experiments on the dumbbell topology

Throughput as function of L when B=100MB/s (limited by latency)





Similar trends in both simulators

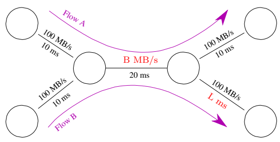
- L < 10 ms ⇒ Flow A gets less bandwidth than B
- L = 10 ms ⇒ Flow A gets as much bandwidth as B
- L > 10 ms ⇒ Flow A gets more bandwidth than B

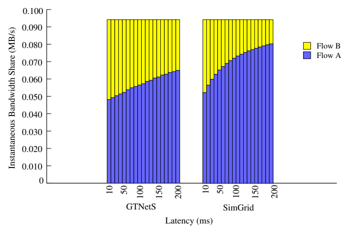
Neglectable error

- $|\varepsilon_{max}| \approx 0.1\%$

Validation experiments on the dumbbell topology

Throughput as function of L when B=0.1MB/s (limited by bandwidth)

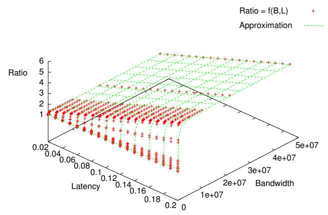




Analysis

- The trend is respected (as L increases share of flow A increases)
- $\overline{|\varepsilon|} \approx 15\%$; $|\varepsilon_{max}| \approx 202\%$
- Model inaccurate or badly instantiated...

Data fitting on the Bandwidth ratio in GTNetS

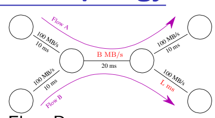


- Ratio $= \frac{\text{Share flow A}}{\text{Share flow B}}$ (in GTNetS)
- Data fitting $= \frac{\sum \left(L_i + \frac{8775}{B_i}\right)}{\sum \left(L_j + \frac{8775}{B_j}\right)}$
- Conclusion: MaxMin needs priorities
  - CM02 use latency only ($w_i = L_i$)
  - bandwidth should be considered

LV08 improvements

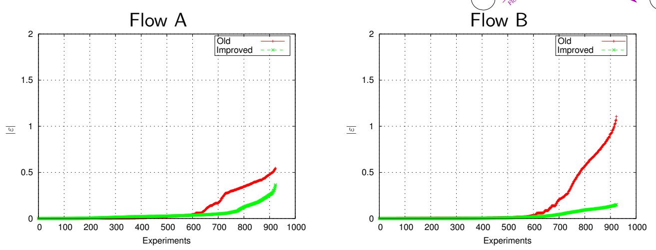
- Max-Min with priorities: $w_i = \sum_{\text{link } k \text{ is used by flow } i} \left(L_k + \frac{8775}{B_k}\right)$
- Improved results:
  - $\overline{|\varepsilon|} \approx 4\%$; $|\varepsilon_{max}| \approx 44\%$

Validation experiments on the dumbbell topology



Summary of all experiences for both flows

Some are bandwidth-limited, some are latency-limited



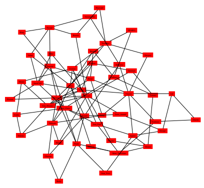
Conclusion

- SimGrid uses an accurate yet fast sharing model
- Improved model is validated against GTNetS
- Accuracy has to be evaluated against more general topologies

Validation experiments on random platforms

Experiments on 160 platforms

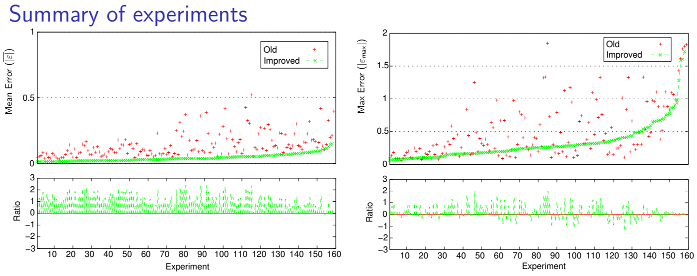
- Platforms generated with BRITE (Waxman model)
- Bandwidth: uniform distribution
  - Homogeneous: B ∈ [100,128] MB/s
  - Heterogeneous: B ∈ [10,128] MB/s
- Latency: $L \in [0; 5]$ ms (euclidian distance)
- Flow size: S=10MB
- #flows: F=150; #nodes: $N \in [50; 200]$
- Four scenarios, ten different flow instantiations



Pedro Vehlo, Arnaud Legrand. *Accuracy Study and Improvement of Network Simulation in the SimGrid Framework.* **Submitted** to SimuTools'09.

Validation experiments on random platforms

Summary of experiments



ratio $= \log(|\varepsilon_{\text{Old}}|) - \log(|\varepsilon_{\text{Improved}}|)$

Interpretation

- Clear improvements of new model
- $\overline{|\varepsilon|} < 0.2$ (*i.e.*, ≈ 22%); $|\varepsilon_{max}|$ still challenging up to 461%

Simulation speed

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

	GTNetS		SimGrid	
# of flows	Simulation time	simulation / simulated	Simulation time	simulation / simulated
10	0.661s	0.856	0.002s	0.002
25	1.651s	1.712	0.008s	0.010
50	3.697s	3.589	0.028s	0.028
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	simulation / simulated	Simulation time	simulation / simulated
10	65s	0.92	0.001s	0.00002
25	163s	1.85	0.008s	0.00010
50	364s	3.89	0.028s	0.00029
100	753s	8.08	0.138s	0.00142
200	1562s	12.59	0.538s	0.00402

- GTNetS execution time linear in both data size and #flows
- SimGrid only depends on #flows

Conclusion

Models of "Grid" Simulators

- Most are overly simplistic (wormhole: slow and inaccurate at best)
- Some are plainly wrong (OptorSim unfortunate sharing policy)

Analytic TCP models not trivial, but possible

- Several models exist in the literature
- They can be implemented efficiently
- SimGrid implements Max-Min fairness, proportional (Vegas & Reno)

SimGrid almost compares to Packet-Level Simulators

- Validity acceptable in a many cases ($\overline{|\varepsilon|} \approx 5\%$ in most cases)
- Validity range clearly delimited
- Maximum error still unacceptable
  - It is often one GTNetS flow that achieves an insignificant throughput
  - Maybe SimGrid is right and GTNetS is wrong?
- SimGrid speedup $\approx 10^3$, GTNetS slowdown up to 10 (ns-2, SSFNet even worse)
- SimGrid execution time depends only on #flows, not data size
- SimGrid can use GTNetS to perform network predictions (for paranoids)

Future Work

Towards Real-World Experiments

- Assess the several models implemented in SimGrid
- Assess Packet-Level simulators themselves
- Use even more realistic platforms: high contention scenarios
- Use more realistic applications: e.g. (NAS benchmark)

Improve the Macrosopic TCP Models in SimGrid

- Decrease maximum error
- Use LV08 by default instead of CM02

Develop New Models

- Compound models (influence of computation load over communications)
- High-speed networks such as quadrics or myrinet
- Model the disks $\left(\lambda + \frac{size}{\beta} \text{ don't seem sufficient}\right)$
- Model multicores

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Platform Instantiation

To use models, one must instantiate them

Key questions

- How can I run my tests on realistic platforms? What is a realistic platform?
- What are platform parameters? What are their values in real platforms?

Sources of platform descriptions

- Manual modeling: define the characteristics with your sysadmins
- Automatic mapping
- Synthetic platform generator

What is a Platform Instance Anyway?

Structural description

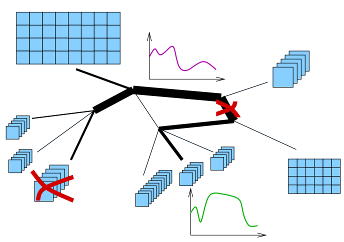
- Hosts list
- Links and interconnexion topology

Peak Performance

- Bandwidth and Latencies
- Processing capacity

Background Conditions

- Load
- Failures



Platform description for SimGrid

Example of XML file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
  <host id="Jacquelin" power="137333000"/>
  <host id="Boivin" power="98095000">
    <prop key="someproperty" value="somevalue"/> <!-- attach arbitrary data to hosts/links -->
  </host>
  <link id="1" bandwidth="3430125" latency="0.000536941"/>
  <route src="Jacquelin" dst="Boivin"><link:ctn id="1"/></route>
  <route src="Boivin" dst="Jacquelin"><link:ctn id="1"/></route>
</platform>
```

- Declare all your hosts, with their computing power other attributes:
  - availability_file: trace file to let the power vary
  - state_file: trace file to specify whether the host is up/down
- Declare all your links, with bandwidth and latency
  - bandwidth_file, latency_file, state_file: trace files
  - sharing_policy $\in$ {shared, fatpipe} (fatpipe $\rightsquigarrow$ no sharing)
- Declare routes from each host to each host (list of links)
- Arbitrary data can be attached to components using the <prop> tag

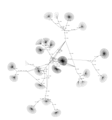
Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

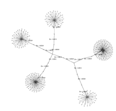
Platform Catalog

Several Existing Platforms Modeled

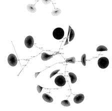
Grid'5000
9 sites, 25 clusters
1,528 hosts



DAS 3
5 clusters
277 hosts



GridPP
18 clusters
7,948 hosts



LCG
113 clusters
44,184 hosts

Files available from the Platform Description Archive
`http://pda.gforge.inria.fr`

(+ tool to extract platform subsets)

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Synthetic Topology Generation

Characterizing Platform Realism (to design a generator)

- Examine real platforms
- Discover principles
- Implement a generator

Topology of the Internet

- Subject of studies in Network Community for years
- Decentralized growth, obeying complex rules and incentives

⤳ Could it have a mathematical structure?

⤳ Could we then have generative models?

Three "generations" of graph generators

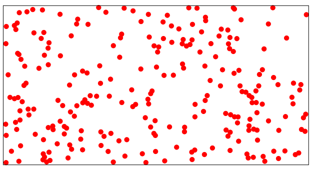
- Random (or flat)
- Structural
- Degree-based

Random Platform Generator

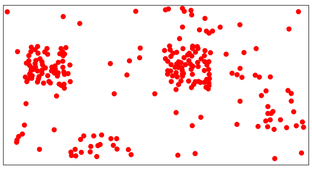
Two-step generators

1. Nodes are placed on on a square (of side c) following a probability law
2. Each couple (u, v) get interconnected with a given probability

1. Node Placement



Uniform



Heavy Tailed

Random Platform Generator

2. Probability for (u, v) get be connected

- Uniform: Uniform probability α (not realist, but simple enough to be popular)
- Exponential: probability $P(u, v) = \alpha e^{-d/(L-d)}$ $\quad 0 < \alpha \leqslant 1$
  - d: Euclidean distance between u and v; $L = c\sqrt{2}$; c side of placement square
  - Amount of edges increases with α
- Waxman: probability $P(u, v) = \alpha e^{-d/(\beta L)}$, $\quad 0 < \alpha, \beta \leqslant 1$
  - Amount of edges increases with α, edge length heterogeneity increases with β

Waxman, *Routing of Multipoint Connections*, IEEE J. on Selected Areas in Comm., 1988.

- Locality-aware: probability $P(u, v) = \begin{cases} \alpha & \text{if } d < L \times r \\ \beta & \text{if } d \geqslant L \times r \end{cases}$

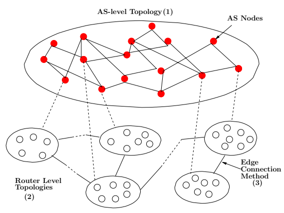
Zegura, Calvert, Donahoo, *A quantitative comparison of graph-based models for Internet topology*, IEEE/ACM Transactions on Networking, 1997.

Structural Topology Generators

Generate the hierarchy explicitly (Top-Down)

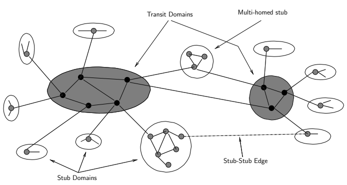
Transit-stub [Zegura *et Al*]

- Starting from a connected graph
- Replace some nodes by connected graphs
- Add some additional edges
- (GT-ITM, BRITE)



N-level [Zegura *et Al*]

- Iterate previous algorithm
- (Tiers, GT-ITM)

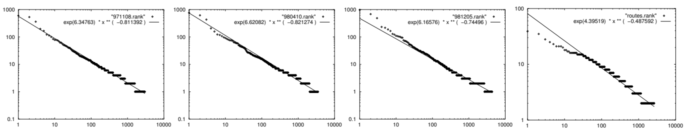


Power-Law : Rank Exponent

Analysis of topology at AS level

- Rank r_v of node v: its index in the order of decreasing degree
- Degree d_v of node v is proportional to its rank r_v to the power of constant $\mathcal{R}$

$$d_v = r_v^{\mathcal{R}} \times k$$



Nov 97 ($\mathcal{R} = 0,81$) — Apr 98 ($\mathcal{R} = 0,82$) — Dec 98 ($\mathcal{R} = 0,74$) — Routers 95 ($\mathcal{R} = 0,48$)

Seem to be necessary condition for topology realism

Faloutsos, Faloutsos, Faloutsos, *On Power-law Relationships of the Internet Topology*, SIGCOMM 1999, p251–262.

Degree-based Topology Generators

Power-laws received a lot of attention recently

- Small-World theory
- Not only in CS, but also in sociology for example

Using this idea for realistic platform generation

- Enforce the power law by construction of the platform

Baràbasi-Albert algorithm

- Incremental growth
- Affinity connexion

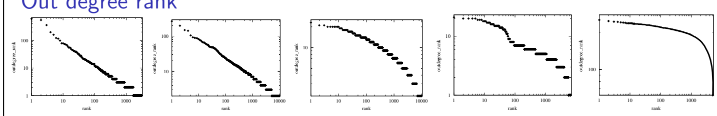
Probability to connect new v to existing u

- Depends on d_u: $\quad P(u, v) = \dfrac{d_u}{\sum_k d_k}$

Barabási and Albert, *Emergence of scaling in random networks*, Science 1999, num 59, p509–512.

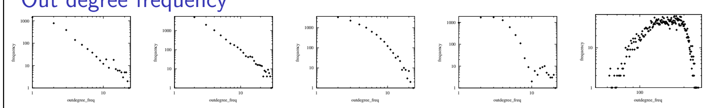
Checking two Power-Laws

Out degree rank



Interdomain 11/97 — Barábasi Albert (BRITE) — Waxman — Transit-Stub (GT-ITM) — GT-ITM

Out degree frequency



Interdomain 11/97 — Barábasi Albert (BRITE) — Waxman — Transit-Stub (GT-ITM) — GT-ITM

- Laws respected by interdomain topology ⤳ seemingly necessary condition
- Baràbasi-Albert performs the best (as expected)
- GT-ITM performs the worst

Power laws discussion

Other power laws? On which measurements?

- Expansion
- Resilience
- Distortion
- Excentricity distribution
- Eigenvalues distribution
- Set cover size, ...

Methodological limits

- Necessary condition ≠ sufficient condition
- Laws observed by Faloutsos brothers are correlated
- They could be irrelevant parameters

Baford, Bestavros, Byers, Crovella, *On the Marginal Utility of Network Topology Measurements*, 1st ACM SIGCOMM Workshop on Internet Measurement, 2001.

- They could even be measurement bias!

Lakhina, Byers, Crovella, Xie, *Sampling Biases in IP Topology Measurements*, INFOCOM'03.

Networks have Power Laws AND structure!

- Cannot afford to trash hierarchical structures just to obey power laws!
- Some projects try to combine both (GridG)

So, Structural or Degree-based Topology Generator?

Observation

- AS-level and router-level have similar characteristics
- Degree-based represent better large-scale properties of the Internet
- Hierarchy seems to arise from degree-based generators

Tangmunarunkit, Govindan, Jamin, Shenker, Willinger, *Network topology generators: Degree-based vs structural*, SIGCOMM'02

Conclusion

- 10,000 nodes platform: Degree-based generators perform better
- 100 nodes platform
  - Power-laws make no sense
  - Structural generators seem more appropriate

Routing still remains to be characterized

- It is known that a multi-hop network route is not always the shortest path

Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*, PhD Thesis UCB, 1997.

- Generators wrongly assume the opposite

Network Performance (labeling graph edges)

We need more than a graph!

- Bandwidth and latency
- Sharing capacity (backplane)

Model Physical Characteristics (Peak Performance+Background)

- Some "models" in topology generators (WAN/LAN/SAN)
- Need to simulate background traffic (no accepted model to generate it)
- Simulation can be very costly

Model End-to-End Performance (Usable Performance)

- Easier way to go
- Some models exist Lee, Stepanek, *On future global grid communication performance*, HCW
- Use real raw measurements (NWS, ...)

Computing Resources (labeling graph vertices)

Situation quite different from network resources:

- Hard to qualify usable performance
- Easy to model peak performance + background conditions

"Ad-hoc" generalization of peak performance

- Look at a real-world platform, *e.g.*, the TeraGrid
- Generate new sites based on existing sites

Statistical modeling (as usual)

- Examine many production resources
- Identify key statistical characteristics
- Come up with a generative/predictive model

Synthetic Clusters

Clusters are classical resource

- What is the "typical" distribution of clusters?

Commodity Cluster synthesizer

- Examined 114 production clusters (10K+ procs)
- Came up with statistical models
  - Linear fit between clock-rate and release-year within a processor family
  - Quadratic fraction of processors released on a given year
- Validated model against a set of 191 clusters (10K+ procs)
- Models allow "extrapolation" for future configurations
- Models implemented in a resource generator

Kee, Casanova, Chien, *Realistic Modeling and Synthesis of Resources for Computational Grids*, Supercomputing 2004.

Background Conditions (workload and resource availability)

Probabilistic Models

- Naive: experimental distributed availability and unavailability intervals
- Weibull distributions:

Nurmi, Brevik, Wolski, *Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments*, EuroPar 2005.

- Models by Feitelson *et Al.*: job inter-arrival times (Gamma), amount of work requested (Hyper-Gamma), number of processors requested: Compounded (2^p, 1, ...)

Feitelson, *Workload Characterization and Modeling Book*, available at `http://www.cs.huji.ac.il/~feit/wlmod/`

Traces

- The Grid Workloads Archive (`http://gwa.ewi.tudelft.nl/pmwiki/`)
- Resource Prediction System Toolkit (RPS) based traces (`http://www.cs.northwestern.edu/~pdinda/LoadTraces`)
- Home-made traces with NWS

Example Synthetic Grid Generation

Generate topology and networks

- Topology: Generate a 5,000 node graph with Tiers
- Latency: Euclidian distance (scaling to obtain the desired network diameter)
- Bandwidth: Set of end-to-end NWS measurements

Generate computational resources

- Pick 30% of the end-points
- Clusters at each end-point: Kee's synthesizer for Year 2008
- Cluster load: Feitelson's model (parameters picked randomly)
- Resource failures: based on the Grid Workloads Archive

All-in-one tools

- GridG

Lu and Dinda, *GridG: Generating Realistic Computational Grids*, Performance Evaluation Review, Vol. 30::4 2003.

- Simulacrum tool

Quinson, Suter, *A Platform Description Archive for Reproducible Simulation Experiments*, **Submitted to** SimuTools'09.

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Automatic Network Mapping

Main Issue of synthetic generators: Realism!

- Solution: Actually map a real platform

Several levels of information (depending on the OSI layer)

- Physical inter-connexion map (wires in the walls)
- Routing infrastructure (path of network packets, from router to switch)
- Application level (focus on effects – bandwidth & latency – not causes)
  Our goal: conduct experiments at application level, not administrating tool

Network Mapping Process: two-step

1. Measurements
2. Reconstruct a graph

Classical Measurements in a Grid Environment?

Use of low-level network protocols (like SNMP or BGP)

- Example: Remos
- Use of SNMP restricted for security reasons (DoS or spying)

Use of traceroute or ping (i.e. on ICMP)

- Examples: TopoMon, Lumeta, IDmaps, Global Network Positioning
- Use of ICMP more and more restricted by admins (for security reasons)

Pathchar

- No network privilege required, but must be root on hosts

⇒ not adapted to Grid settings

Measurements must be at application-level (no privilege)

Solutions relying on application-level measurements

NWS (Network Weather Service – UCSB)

- *De facto* standard (used in Globus, DIET, NINF) to gather info on network
- Reports bandwidth, latency, CPU availability, and future trends
- Only quantitative values, no topological information
  (but one can label a big clique with NWS-provided values)

ENV (Effective Network View – UCSD)

- Use interference measurements to build a tree representation

ECO (Efficient Collective Communication – CMU)

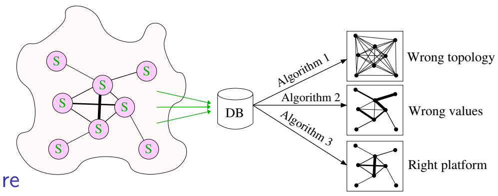
- Use application-level measurements to optimize collective communications
- Should be generalized

Existing reconstruction algorithms

- Cliques (NWS, ECO) or trees (ENV, Classical latency clustering)

ALNeM (Application-Level Network Mapper)

- Long-term goal: be a tool providing topology to network-aware applications
- Short-term goal: allow the study of network mapping algorithms



Architecture

- Lightweight distributed measurement infrastructure (collection of sensors)
- MySQL measurement database
- Topology builder, with several reconstruction algorithms

Eyraud-Dubois, Legrand, Quinson, Vivien, *A First Step Towards Automatically Building Network Representations*, EuroPar'07.

Reconstruction algorithms

Basic algorithms

- Clique: Connect all pairs of nodes, label with measured values
- Maximum Bandwidth Spanning Tree and Minimum Latency Spanning Tree

Improved Spanning Tree

- Real platforms are not trees, BwTree and LatTree miss edges

⇝ Add edges to spanning trees to improve predictions

Aggregation

- Grow a set of connected nodes
- For each new one, connect it to already chosen ones to improve predictions

Evaluation methodology

- Goal: Quantify similarity between initial and reconstructed platforms
- Running *in situ*: beware of experimental bias!
  - Reconstructed platform doesn't exist in the real world
  - ⇒ cannot compare measurements on both platforms
  - ⇒ hard to assess quality of reconstruction algorithms on real platforms
- Testing on simulator: both initial and reconstructed platforms are simulated

Several evaluation metrics

1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:

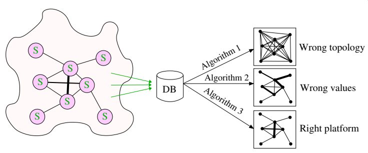
$$Interf\left((a,b),(c,d)\right) = 1 \text{ iff } \frac{BW(a \rightarrow b)}{BW(a \rightarrow b \parallel c \rightarrow d)} \approx 2$$

3. Compare application running times (application-level)

	Comm. schema	// comm	# steps
Token-ring	Ring	No	1
Broadcast	Tree	No	1
All2All	Clique	Yes	1
Parallel Matrix Multiplication	2D	Yes	$\sqrt{procs}$

Evaluation methodology

Apply all evaluations on all reconstructions for several platforms



Measurements

- Bandwidth matrix
- Latency matrix

Algorithms

- Clique
- BW/Lat Spanning Tree
- Improved BW/Lat Tree
- Aggregate

Evaluation criteria

- End-to-end meas.
- Interference count
- Application-level

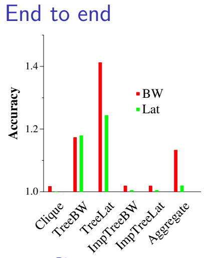
Development on simulator, creating a real tool for real platforms

- Measurement sensors implemented using GRAS:
  Same code running either on top of SimGrid, or in situ (more to come)
- ALNeM usable in situ, presumably with same predictive quality

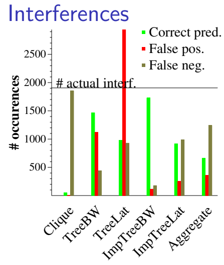
Experiments on simulator: Renater platform

- *Real* platform built manually (real measurements + admin feedback)

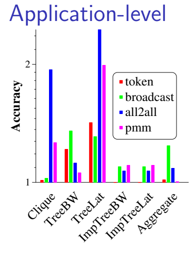
End to end



Interferences



Application-level

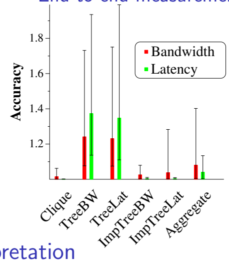


- Clique:
  - Very good for end-to-end (of course)
  - No contention captured ⇝ missing interference ⇝ bad predictions
- Spanning Trees: missing links ⇝ bad predictions
  (over-estimates latency, under-estimates bandwidth, false positive interference)
- Improved Spanning Trees have good predictive power
- Aggregate accuracy discutable

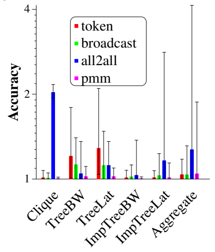
Experiments on simulator: GridG platforms

- GridG is a synthetic platform generator [Lu, Dinda – SuperComputing03]
  Generates *realistic* platforms
- Experiment: 40 platforms (60 hosts – default GridG parameters)

End to end measurements



Application-level measurements



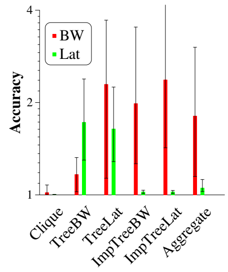
Interpretation

- Naive algorithms lead to poor results
- Improved trees yield good reconstructions
  - ImpTreeBW error ≈ 3% for all2all (worst case)

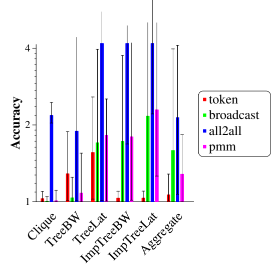
Adding routers to the picture

- New set of experiments: only *leaf* nodes run the measurement processes

End to end measurements



Application-level measurements



Interpretation

- None of the proposed heuristic is satisfactory
- Future work: improve this!

Conclusions about ALNeM

Reconstruction algorithm evaluation from application POV

- Several quality criteria: similarity of end-to-end, interferences, application timings
- Runs on simulator or *in-situ* thanks to GRAS (& SimGrid)
  (successfully reconstructed real platforms, but quality assessment very hard)

Classical algorithms are not satisfactory

- Spanning trees: miss edges, leading to performance under-estimation
- Cliques: do not capture any existing interference
- Improving spanning trees yields much better results (specially ImpTreeBW)
- Still problems with internal routers

Future work

- Other measurements from the sensors (new inputs to algorithms)
  Interference (but very expensive to acquire); Packet gap and back-to-back packets
- Method based on successive refinements
  1. Spanning tree as first approximation
  2. Refinement by adding some missing links
  3. Some (not all) interference measurements to double-check the result

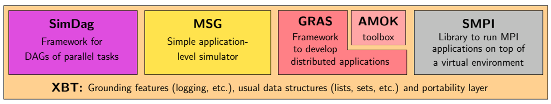
Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - 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: model applications as DAG of (parallel) tasks
- MSG: model applications as Concurrent Sequential Processes
- 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)*
- XBT: grounding toolbox

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

Argh! Do I really have to code in C?!

No, not necessary

- Some bindings exist: Java bindings to the MSG interface (new in v3.3)
- More bindings planned:
  - C++, Python, and any scripting language
  - SimDag interface

Well, sometimes yes, but...

- SimGrid itself is written from C for speed and portability (no dependency)
- All components naturally usable from C (most of them only accessible from C)
- XBT eases some difficulties of C
  - Full-featured logs (similar to log4j), Exception support (in ANSI C)
  - Popular abstract data types (dynamic array, hash tables, ...)
  - Easy string manipulation, Configuration, Unit testing, ...

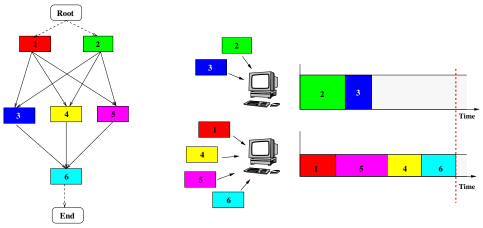
What about portability?

- Regularly tested under: Linux (x86, amd64), *Windows* and MacOSX
- Supposed to work under any other Unix system (including AIX and Solaris)

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

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)
   ⇝ Compute the makespan

The SimDag interface

DAG creation

- Creating tasks: `SD_task_create(name, data)`
- Creating dependencies: `SD_task_dependency_{add/remove}(src,dst)`

Scheduling tasks

- `SD_task_schedule(task, workstation_number, *workstation_list, double *comp_amount, double *comm_amount, double rate)`
  - Tasks are parallel by default; simply put workstation_number to 1 if not
  - Communications are regular tasks, comm_amount is a matrix
  - Both computation and communication in same task possible
  - rate: To slow down non-CPU (resp. non-network) bound applications
- `SD_task_unschedule, SD_task_get_start_time`

Running the simulation

- `SD_simulate(double how_long)` (how_long $< 0 \leadsto$ until the end)
- `SD_task_{watch/unwatch}`: simulation stops as soon as task's state changes

Full API in the doxygen-generated documentation

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - Motivations, Concepts and Example of Use
    - Java bindings
    - A Glance at SimGrid Internals
    - Performance Results
  - GRAS: Developing and Debugging Real Applications
- Conclusion

MSG: Heuristics for Concurrent Sequential Processes

(historical) Motivation

- Centralized scheduling does not scale
- SimDag (and its predecessor) 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
  set of functions + XML deployment file for arguments
- **Task:** amount of work to do and of data to exchange
  - `MSG_task_create(name, compute_duration, message_size, void *data)`
  - Communication: `MSG_task_{put,get}, MSG_task_Iprobe`
  - Execution: `MSG_task_execute`
    `MSG_process_sleep, MSG_process_{suspend,resume}`
- Host: location on which agents execute
- Mailbox: similar to MPI tags

The MSG master/workers example: the worker

The master has a large number of tasks to dispatch to its workers for execution

```
int worker(int argc, char *argv[ ]) {
  m_task_t task;                 int errcode;
  int id = atoi(argv[1]);
  char mailbox[80];

  sprintf(mailbox,"worker-%d",id);

  while(1) {
    errcode = MSG_task_receive(&task, mailbox);
    xbt_assert0(errcode == MSG_OK, "MSG_task_get failed");

    if (!strcmp(MSG_task_get_name(task),"finalize")) {
      MSG_task_destroy(task);
      break;
    }

    INFO1("Processing '%s'", MSG_task_get_name(task));
    MSG_task_execute(task);
    INFO1("'%s' done", MSG_task_get_name(task));
    MSG_task_destroy(task);
  }

  INFO0("I'm done. See you!");
  return 0;
}
```


The MSG master/workers example: the master

```
int master(int argc, char *argv[ ]) {
 int number_of_tasks = atoi(argv[1]);      double task_comp_size = atof(argv[2]);
 double task_comm_size = atof(argv[3]);    int workers_count = atoi(argv[4]);
 char mailbox[80];                         char buff[64];
 int i;

 /* Dispatching (dumb round-robin algorithm) */
 for (i = 0; i < number_of_tasks; i++) {
   sprintf(buff, "Task_%d", i);
   task = MSG_task_create(sprintf_buffer, task_comp_size, task_comm_size, NULL);
   sprintf(mailbox,"worker-%d",i % workers_count);
   INFO2("Sending %s¨ to mailbox %s", task->name, mailbox);
   MSG_task_send(task, mailbox);
 }

 /* Send finalization message to workers */
 INFO0("All tasks dispatched. Let's stop workers");
 for (i = 0; i < workers_count; i++)
  MSG_task_put(MSG_task_create("finalize", 0, 0, 0), workers[i], 12);

 INFO0("Goodbye now!"); return 0;
}
```


The MSG master/workers example: deployment file

Specifying which agent must be run on which host, and with which arguments

XML deployment file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">

  <!-- The master process (with some arguments) -->
  <process host="Tremblay" function="master">
     <argument value="6"/>         <!-- Number of tasks -->
     <argument value="50000000"/> <!-- Computation size of tasks -->
     <argument value="1000000"/>  <!-- Communication size of tasks -->
     <argument value="3"/>         <!-- Number of workers -->
  </process>

  <!-- The worker process (argument: mailbox number to use) -->
  <process host="Jupiter" function="worker"><argument value="0"/></process>
  <process host="Fafard"  function="worker"><argument value="1"/></process>
  <process host="Ginette" function="worker"><argument value="2"/></process>

</platform>
```


The MSG master/workers example: the main()

Putting things together

```
int main(int argc, char *argv[ ]) {
  /* Declare all existing agent, binding their name to their function */
  MSG_function_register("master", &master);
  MSG_function_register("worker", &worker);

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

  /* Launch the simulation (until its end) */
  MSG_main();

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

The MSG master/workers example: raw output

```
[Tremblay:master:(1) 0.000000] [example/INFO] Got 3 workers and 6 tasks to process
[Tremblay:master:(1) 0.000000] [example/INFO] Sending 'Task_0' to 'worker-0'
[Tremblay:master:(1) 0.147613] [example/INFO] Sending 'Task_1' to 'worker-1'
[Jupiter:worker:(2) 0.147613] [example/INFO] Processing 'Task_0'
[Tremblay:master:(1) 0.347192] [example/INFO] Sending 'Task_2' to 'worker-2'
[Fafard:worker:(3) 0.347192] [example/INFO] Processing 'Task_1'
[Tremblay:master:(1) 0.475692] [example/INFO] Sending 'Task_3' to 'worker-0'
[Ginette:worker:(4) 0.475692] [example/INFO] Processing 'Task_2'
[Jupiter:worker:(2) 0.802956] [example/INFO] 'Task_0' done
[Tremblay:master:(1) 0.950569] [example/INFO] Sending 'Task_4' to 'worker-1'
[Jupiter:worker:(2) 0.950569] [example/INFO] Processing 'Task_3'
[Fafard:worker:(3) 1.002534] [example/INFO] 'Task_1' done
[Tremblay:master:(1) 1.202113] [example/INFO] Sending 'Task_5' to 'worker-2'
[Fafard:worker:(3) 1.202113] [example/INFO] Processing 'Task_4'
[Ginette:worker:(4) 1.506790] [example/INFO] 'Task_2' done
[Jupiter:worker:(2) 1.605911] [example/INFO] 'Task_3' done
[Tremblay:master:(1) 1.635290] [example/INFO] All tasks dispatched. Let's stop workers.
[Ginette:worker:(4) 1.635290] [example/INFO] Processing 'Task_5'
[Jupiter:worker:(2) 1.636752] [example/INFO] I'm done. See you!
[Fafard:worker:(3) 1.857455] [example/INFO] 'Task_4' done
[Fafard:worker:(3) 1.859431] [example/INFO] I'm done. See you!
[Ginette:worker:(4) 2.666388] [example/INFO] 'Task_5' done
[Tremblay:master:(1) 2.667660] [example/INFO] Goodbye now!
[Ginette:worker:(4) 2.667660] [example/INFO] I'm done. See you!
[2.667660] [example/INFO] Simulation time 2.66766
```


The MSG master/workers example: colorized output

```
$ ./my_simulator | MSG_visualization/colorize.pl
[    0.000][  Tremblay:master   ] Got 3 workers and 6 tasks to process
[    0.000][  Tremblay:master   ] Sending 'Task_0' to 'worker-0'
[    0.148][  Tremblay:master   ] Sending 'Task_1' to 'worker-1'
[    0.148][   Jupiter:worker   ] Processing 'Task_0'
[    0.347][  Tremblay:master   ] Sending 'Task_2' to 'worker-2'
[    0.347][    Fafard:worker   ] Processing 'Task_1'
[    0.476][  Tremblay:master   ] Sending 'Task_3' to 'worker-0'
[    0.476][   Ginette:worker   ] Processing 'Task_2'
[    0.803][   Jupiter:worker   ] 'Task_0' done
[    0.951][  Tremblay:master   ] Sending 'Task_4' to 'worker-1'
[    0.951][   Jupiter:worker   ] Processing 'Task_3'
[    1.003][    Fafard:worker   ] 'Task_1' done
[    1.202][  Tremblay:master   ] Sending 'Task_5' to 'worker-2'
[    1.202][    Fafard:worker   ] Processing 'Task_4'
[    1.507][   Ginette:worker   ] 'Task_2' done
[    1.606][   Jupiter:worker   ] 'Task_3' done
[    1.635][  Tremblay:master   ] All tasks dispatched. Let's stop workers.
[    1.635][   Ginette:worker   ] Processing 'Task_5'
[    1.637][   Jupiter:worker   ] I'm done. See you!
[    1.857][    Fafard:worker   ] 'Task_4' done
[    1.859][    Fafard:worker   ] I'm done. See you!
[    2.666][   Ginette:worker   ] 'Task_5' done
[    2.668][  Tremblay:master   ] Goodbye now!
[    2.668][   Ginette:worker   ] I'm done. See you!
[    2.668][                    ] Simulation time 2.66766
```


Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - Motivations, Concepts and Example of Use
    - Java bindings
    - A Glance at SimGrid Internals
    - Performance Results
  - GRAS: Developing and Debugging Real Applications
- Conclusion

MSG bindings for Java: master/workers example

```
import simgrid.msg.*;
public class BasicTask extends simgrid.msg.Task {
   public BasicTask(String name, double computeDuration, double messageSize)
                                                            throws JniException {
      super(name, computeDuration, messageSize);
   }
}
public class FinalizeTask extends simgrid.msg.Task {
   public FinalizeTask() throws JniException {
      super("finalize",0,0);
   }
}
public class Worker extends simgrid.msg.Process {
   public void main(String[ ] args) throws JniException, NativeException {
      String id = args[0];

      while (true) {
         Task t = Task.receive("worker-" + id);
         if (t instanceof FinalizeTask)
            break;
         BasicTask task = (BasicTask)t;
         Msg.info("Processing '" + task.getName() + "'");
         task.execute();
         Msg.info("'" + task.getName() + "' done ");
      }
      Msg.info("Received Finalize. I'm done. See you!");
   }
}
```


MSG bindings for Java: master/workers example

```
import simgrid.msg.*;
public class Master extends simgrid.msg.Process {
   public void main(String[ ] args) throws JniException, NativeException {
      int numberOfTasks = Integer.valueOf(args[0]).intValue();
      double taskComputeSize = Double.valueOf(args[1]).doubleValue();
      double taskCommunicateSize = Double.valueOf(args[2]).doubleValue();
      int workerCount = Integer.valueOf(args[3]).intValue();

      Msg.info("Got "+  workerCount + " workers and " + numberOfTasks + " tasks.");

      for (int i = 0; i < numberOfTasks; i++) {
         BasicTask task = new BasicTask("Task_" + i ,taskComputeSize,taskCommunicateSize);
         task.send("worker-" + (i % workerCount));

         Msg.info("Send completed for the task " + task.getName() +
                  " on the mailbox 'worker-" + (i % workerCount) +  "'");
      }
      Msg.info("Goodbye now!");
   }
}
```


MSG bindings for Java: master/workers example

Rest of the story

- XML files (platform, deployment) not modified
- No need for a main() function glueing things together
  - Java introspection mecanism used for this
  - `simgrid.msg.Msg` contains an adapted `main()` function
  - Name of XML files must be passed as command-line argument
- Output very similar too

What about performance loss?

tasks \ workers		100	500	1,000	5,000	10,000
1,000	native	.16	.19	.21	.42	0.74
	java	.41	.59	.94	7.6	27.
10,000	native	.48	.52	.54	.83	1.1
	java	1.6	1.9	2.38	13.	40.
100,000	native	3.7	3.8	4.0	4.4	4.5
	java	14.	13.	15.	29.	77.
1,000,000	native	36.	37.	38.	41.	40.
	java	121.	130.	134.	163.	200.

- Small platforms: ok
- Larger ones: not quite. . .

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - Motivations, Concepts and Example of Use
    - Java bindings
    - A Glance at SimGrid Internals
    - Performance Results
  - GRAS: Developing and Debugging Real Applications
- Conclusion

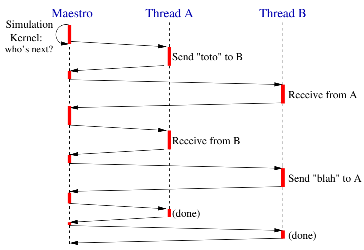
Implementation of CSPs on top of simulation kernel

Idea

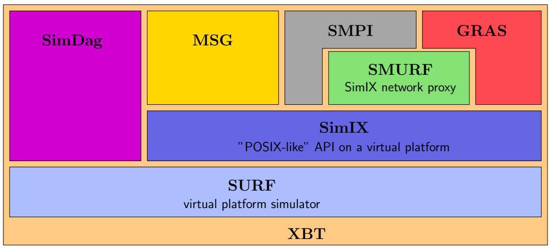
- Each process is implemented in a thread
- Blocking actions (execution and communication) reported into kernel
- A *maestro* thread unlocks the runnable threads (when action done)

Example

- Thread A:
  - Send "toto" to B
  - Receive something from B
- Thread B:
  - Receive something from A
  - Send "blah" to A
- Maestro schedules threads
  Order given by simulation kernel
- Mutually exclusive execution
  (don't fear)



A Glance at SimGrid Internals



- SURF: Simulation kernel, grounding simulation
  Contains all the models (uses GTNetS on need)
- SimIX: Eases the writting of user APIs based on CSPs
  Provided semantic: threads, mutexes and conditions on top of simulator
- SMURF: Allows to distribute the simulation over a cluster *(under development)*
  Not for speed but for memory limit (at least for now)

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - Motivations, Concepts and Example of Use
    - Java bindings
    - A Glance at SimGrid Internals
    - Performance Results
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Some Performance Results

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

⇒ at most 10,922 user processes

†: out of memory

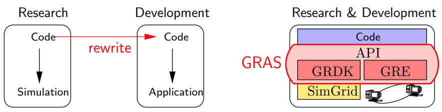
Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
    - Motivation and project goals
    - Functionalities
    - Experimental evaluation (performance and simplicity)
    - Conclusion and Perspectives
- Conclusion

Goals of the GRAS project (Grid Reality And Simulation)

Ease development of large-scale distributed apps

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 ≠ prototype)
  - Performance concern: efficient communication of structured data
    How: Efficient wire protocol (avoid data conversion)
  - Portability concern: because of grid heterogeneity
    How: ANSI C + autoconf + no dependency

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
    - Motivation and project goals
    - Functionalities
    - Experimental evaluation (performance and simplicity)
    - Conclusion and Perspectives
- Conclusion

Main concepts of the GRAS API

Agents (acting entities)

- Code (C function)
- Private data
- Location (hosting computer)

Sockets (communication endpoints)

- Server socket: to receive messages
- Client socket: to contact a server (and receive answers)

Messages (what gets exchanged between agents)

- Semantic: Message type
- Payload described by data type description (fixed for a given type)

Callbacks (code to execute when a message is received)

- Also possible to explicitly wait for given messages

Emulation and Virtualization

Same code runs **without modification** both in simulation and *in situ*

- In simulation, agents run as threads within a single process
- In situ, each agent runs within its own process

⇒ Agents are threads, which can run as separate processes

Emulation issues

- How to get the process sleeping? How to get the current time?
  - System calls are *virtualized*: `gras_os_time`; `gras_os_sleep`
- How to report computation time into the simulator?
  - Asked explicitly by user, using provided macros
  - Time to report can be benchmarked automatically
- What about global data?
  - Agent status placed in a specific structure, ad-hoc manipulation API

Example of code: ping-pong (1/2)

Code common to client and server

```
#include "gras.h"
XBT_LOG_NEW_DEFAULT_CATEGORY(test,"Messages specific to this example");
static void register_messages(void) {
  gras_msgtype_declare("ping", gras_datadesc_by_name("int"));
  gras_msgtype_declare("pong", gras_datadesc_by_name("int"));
}
```

Client code

```
int client(int argc,char *argv[ ]) {
  gras_socket_t peer=NULL, from ;
  int ping=1234, pong;

  gras_init(&argc, argv);
  gras_os_sleep(1); /* Wait for the server startup */
  peer=gras_socket_client("127.0.0.1",4000);
  register_messages();

  gras_msg_send(peer, "ping", &ping);
  INFO3("PING(%d) -> %s:%d",ping, gras_socket_peer_name(peer), gras_socket_peer_port(peer));
  gras_msg_wait(6000,"pong",&from,&pong);

  gras_exit();
  return 0;
}
```


Example of code: ping-pong (2/2)

Server code

```
typedef struct { /* Global private data */
  int endcondition;
} server_data_t;
int server (int argc,char *argv[ ]) {
  server_data_t *globals;
  gras_init(&argc,argv);
  globals = gras_userdata_new(server_data_t);
  globals->endcondition=0;
  gras_socket_server(4000);
  register_messages();
  gras_cb_register("ping", &server_cb_ping_handler);

  while (!globals->endcondition) { /* Handle messages until our state change */
    gras_msg_handle(600.0);       /* Actually, one ping is enough for that */
  }
  free(globals);  gras_exit();  return 0;
}
int server_cb_ping_handler(gras_msg_cb_ctx_t ctx, void *payload_data) {
  server_data_t *globals = (server_data_t*)gras_userdata_get(); /* Get the globals */
  globals->endcondition = 1;

  int msg = *(int*) payload_data;                        /* What's the content? */
  gras_socket_t expeditor = gras_msg_cb_ctx_from(ctx);   /* Who sent it?*/
  /* Send data back as payload of a pong message to the ping's expeditor */
  gras_msg_send(expeditor, "pong", &msg);
  return 0;
}
```


Exchanging structured data

GRAS wire protocol: NDR (Native Data Representation)

Avoid data conversion when possible:

- Sender writes data on socket as they are in memory
- If receiver's architecture does match, no conversion
- Receiver able to convert from any architecture

GRAS message payload can be any valid C type

- Structure, enumeration, array, pointer, ...
- Classical garbage collection algorithm to deep-copy it
- Cycles in pointed structures detected & recreated

Describing a data type to GRAS

Manual description (excerpt)

```
gras_datadesc_type_t gras_datadesc_struct(name);
gras_datadesc_struct_append(struct_type,name,field_type);
gras_datadesc_struct_close(struct_type);
```

Automatic description of vector

```
GRAS_DEFINE_TYPE(s_vect,
 struct s_vect {
  int cnt;
  double*data GRAS_ANNOTE(size,cnt);
 }
);
```

C declaration stored into a `char*` variable to be parsed at runtime

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
    - Motivation and project goals
    - Functionalities
    - Experimental evaluation (performance and simplicity)
    - Conclusion and Perspectives
- Conclusion

Assessing communication performance

Only communication performance studied since computation are not mediated

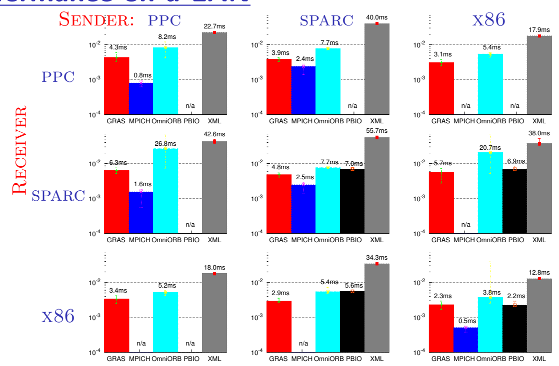
- Experiment: timing ping-pong of structured data (a message of Pastry)

```
typedef struct {
  int id, row_count;
  double time_sent;
  row_t *rows;
  int leaves[MAX_LEAFSET];
} welcome_msg_t;
```

```
typedef struct {
  int which_row;
  int row[COLS][MAX_ROUTESET];
} row_t;
```

- Tested solutions
  - GRAS
  - PBIO (uses NDR)
  - OmniORB (classical CORBA solution)
  - MPICH (classical MPI solution)
  - XML (Expat parser + handcrafted communication)
- Platform: x86, PPC, sparc (all under Linux)

Performance on a LAN



- MPICH twice as fast as GRAS, but cannot mix little- and big-endian Linux
- PBIO broken on PPC
- XML much slower (extra conversions + verbose wire encoding)

GRAS is the better compromise between performance and portability

Assessing API simplicity

Experiment: ran code complexity measurements on code for previous experiment

	GRAS	MPICH	PBIO	OmniORB	XML
McCabe Cyclomatic Complexity	8	10	10	12	35
Number of lines of code	48	65	84	92	150

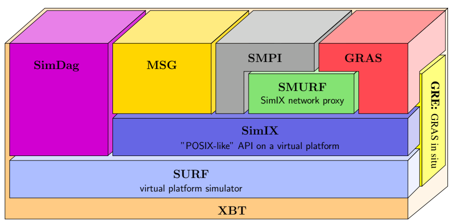
Results discussion

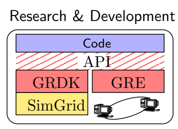
- XML complexity may be artefact of Expat parser (but fastest)
- MPICH: manual marshaling/unmarshalling
- PBIO: automatic marshaling, but manual type description
- OmniORB: automatic marshaling, IDL as type description
- GRAS: automatic marshaling & type description (IDL is C)

Conclusion

GRAS is the least demanding solution from developer perspective

Conclusion: GRAS eases infrastructure development





With GRAS

GRDK: Grid Research & Development Kit

- API for (explicitly) distributed applications
- Study applications in the comfort of the simulator

GRE: Grid Runtime Environment

- Efficient: twice as slow as MPICH, faster than OmniORB, PBIO, XML
- Portable: Linux (11 CPU archs); *Windows*; Mac OS X; Solaris; IRIX; AIX
- Simple and convenient:
  - API simpler than classical communication libraries (+XBT tools)
  - Easy to deploy: C ANSI; no dependency; autotools; <400kb

GRAS perspectives

Future work on GRAS

- Performance: type precompilation, communication taming and compression
- GRASPE (GRAS Platform Expender) for automatic deployment
- Model-checking as third mode along with simulation and in-situ execution

Ongoing applications

- Comparison of P2P protocols (Pastry, Chord, etc)
- Use emulation mode to validate SimGrid models
- Network mapper (ALNeM): capture platform descriptions for simulator
- Large scale mutual exclusion service

Future applications

- Platform monitoring tool (bandwidth and latency)
- Group communications & RPC; Application-level routing; *etc.*

Agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
  - Tools for Experimentations in Large-Scale Distributed Systems
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
  - Experimental Validation of the Simulation Models
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
  - GRAS: Developing and Debugging Real Applications
- Conclusion

Conclusions on Distributed Systems Research

Research on Large-Scale Distributed Systems

- Reflexion about common methodologies needed (reproductible results needed)
- Purely theoritical works limited (simplistic settings $\rightsquigarrow$ NP-complete problems)
- Real-world experiments time and labor consuming; limited representativity
- Simulation appealing, if results remain validated

Simulating Large-Scale Distributed Systems

- Packet-level simulators too slow for large scale studies
- Large amount of ad-hoc simulators, but discutable validity
- Coarse-grain modelization of TCP flows possible (cf. networking community)
- Model instantiation (platform mapping or generation) remains challenging

SimGrid provides interesting models

- Implements non-trivial coarse-grain models for resources and sharing
- Validity results encouraging with regard to packet-level simulators
- Several orders of magnitude faster than packet-level simulators
- 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

- Declare tasks, their precedences, schedule them on resource, get the makespan

MSG: Comparing Heuristics for Concurrent Sequential Processes

- Declare independent agents running a given function on an host
- Let them exchange and execute tasks
- Easy interface, rapid prototyping
- New in SimGrid v3.3: Java bindings for MSG

GRAS: Developing and Debugging Real Applications

- *Develop once, run in simulation or in situ* (debug; test on non-existing platforms)
- Resulting application twice slower than MPICH, faster than omniorb
- Highly portable and easy to deploy

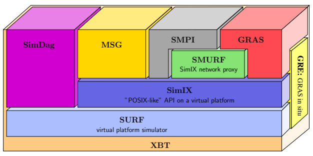
Other interfaces comming

- SMPI: Simulate MPI applications
- BSP model, OpenMP?

SimGrid is an active and exciting project

Future Plans

- Improve usability
  (statistics tools, campaign management)
- Extreme Scalability for P2P
- Model-checking of GRAS applications
- Emulation solution *à la* MicroGrid



Large community

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

- 130 subscribers to the user mailling list (40 to -devel)
- 40 scientific publications using the tool for their experiments
  - 15 co-signed by one of the core-team members
  - 25 purely external
- LGPL, 120,000 lines of code (half for examples and regression tests)
- Examples, documentation and tutorials on the web page

Use it in your works!

Detailed agenda

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
  - Main Methodological Approaches
    - Real-world experiments
    - Simulation
  - Tools for Experimentations in Large-Scale Distributed Systems
    - Possible designs
    - Experimentation platforms: Grid'5000 and PlanetLab
    - Emulators: ModelNet and MicroGrid
    - Packet-level Simulators: ns-2, SSFNet and GTNetS
    - Ad-hoc simulators: ChicagoSim, OptorSim, GridSim, ...
    - Peer to peer simulators
    - SimGrid
- Resource Models in SimGrid
  - Analytic Models Underlying SimGrid
    - Modeling a Single Resource
    - Multi-hop Networks
    - Resource Sharing
  - Experimental Validation of the Simulation Models
    - Single link
    - Dumbbell
    - Random platforms
    - Simulation speed
- Platform Instanciation
  - Platform Catalog
  - Synthetic Topologies
  - Topology Mapping
- Using SimGrid for Practical Grid Experiments
  - Overview of the SimGrid Components
  - SimDag: Comparing Scheduling Heuristics for DAGs
  - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - Motivations, Concepts and Example of Use
    - Java bindings
    - A Glance at SimGrid Internals
    - Performance Results
  - GRAS: Developing and Debugging Real Applications
    - Motivation and project goals
    - Functionalities
    - Experimental evaluation (performance and simplicity)
    - Conclusion and Perspectives
- Conclusion