Scientific Computation Applications

Classical Approaches in science and engineering
1. Theoretical work: equations on a board
2. Experimental study on an scientific instrument

That’s not always desirable (or even possible)
- Some phenomenons are intractable theoretically
- Experiments too expensive, difficult, slow, dangerous

The third scientific way: Computational Science
3. Study in silico using computers
   - Modeling / Simulation of the phenomenon or data-mining
   - High Performance Computing Systems

These systems deserve very advanced analysis
- Their debugging and tuning are technically difficult
- Their use induce high methodological challenges
- Science of the in silico science

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: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications

- The SimGrid Project
  - User Interface(s)
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
  - Associated Tools

- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages
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

▶ In vivo: Direct experimentation
▶ In silico: Simulation
▶ In vitro: Emulation

Outline

Experiments for Large-Scale Distributed Systems Research
Methodological Issues
Main Methodological Approaches: In Vivo, In Silico, In Vitro
  In vivo approach (direct experimentation)
  In silico approach (simulation)
  In vitro approach (emulation)
Existing evaluation tools for HPC ideas / applications

⇒ The SimGrid Project
   User Interface(s)
   Models underlying the SimGrid Framework
   SimGrid Evaluation
   Associated Tools

Conclusions
   Tutorial Recap
   Going Further: Experiment planning and Open Science
   Take-home Messages

In vivo approach to HPC experiments (direct experiment)

○ 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 unrepeateable
  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!

Example of Tools for Direct Experimentation

▶ Principle: Real applications, controlled environment
▶ Challenges: Hard and long. Experimental control? Reproducibility?

Grid’5000 project: a scientific instrument for the HPC

▶ Instrument for research in computer science (deploy your own OS)
▶ 9 sites, 1500 nodes (3000 cpus, 4000 cores); dedicated 10Gb links

Other existing platforms

▶ PlanetLab: No experimental control ⇒ no reproducibility
▶ Production Platforms (EGEE): must use provided middleware
▶ FutureGrid: future US experimental platform loosely inspired from Grid'5000
**In silico approach to HPC experiments (simulation)**

- Simulation solves some difficulties raised by in vivo experiments
  - 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)
  - Instaniciability: 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
  ⇒ 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
  ⇒ 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 → 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)
### In silico approach to HPC experiments (simulation)
- **Principle:** Prototypes of applications, models of platforms
- **Challenges:** Get realistic results (experimental bias)

**SimGrid:** Generic simulation framework for distributed applications
- Scalable (time and memory), modular, portable. +70 publications.

**Other existing tools**
- Large amount of existing simulator for distributed platforms:
  - GridSim, ChicSim, GES; P2PSim, PlanetSim, PeerSim; ns-2, GTNetS.
- Few are really usable: Diffusion, Software Quality Assurance, Long-term availability
- No other study the validity, the induced experimental bias

### In vitro approach to HPC experiments (emulation)
- **Principle:** Injecting load on real systems for the experimental control
  \[ \approx \text{Slow platform down to put it in wanted experimental conditions} \]
- **Challenges:** Get realistic results, tool stack complex to deploy and use

**Wrekavoc:** Applicative emulator
- Emulates CPU and network
- Homogeneous or Heterogeneous platforms

**Other existing tools**
- Network emulation: ModelNet, DummyNet, ... Tools rather mature, but limited to network
- Applicative emulation: MicroGrid, eWan, Emulab Rarely (never?) used outside the lab where they were created

---

### Experiments for Large-Scale Distributed Systems Research

**In silico approach to HPC experiments (simulation)**
- **Principle:** Prototypes of applications, models of platforms
- **Challenges:** Get realistic results (experimental bias)

**SimGrid:** Generic simulation framework for distributed applications
- Scalable (time and memory), modular, portable. +70 publications.

**Other existing tools**
- Large amount of existing simulator for distributed platforms:
  - GridSim, ChicSim, GES; P2PSim, PlanetSim, PeerSim; ns-2, GTNetS.
- Few are really usable: Diffusion, Software Quality Assurance, Long-term availability
- No other study the validity, the induced experimental bias

**Existing evaluation tools for HPC ideas/applications**

<table>
<thead>
<tr>
<th>Tool</th>
<th>CPU</th>
<th>Use</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
<td>&lt;5000</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
<td>dozens</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
<td>hundreds</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>line d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
<td>&lt;1000</td>
</tr>
<tr>
<td>GTENetS</td>
<td>-</td>
<td>-</td>
<td>line d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>&lt;100000</td>
</tr>
<tr>
<td>P2PSim</td>
<td>-</td>
<td>-</td>
<td>cste time</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>100000</td>
</tr>
<tr>
<td>PlanetSim</td>
<td>-</td>
<td>-</td>
<td>state machine</td>
<td>Java</td>
<td>controlled</td>
<td>1000000</td>
<td></td>
</tr>
<tr>
<td>ChicSim</td>
<td>coarse d.e.</td>
<td>-</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>C</td>
<td>controlled</td>
<td>thousands</td>
</tr>
<tr>
<td>OptSim</td>
<td>coarse d.e.</td>
<td>amount</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 100</td>
</tr>
<tr>
<td>GridSim</td>
<td>coarse d.e.</td>
<td>math</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
<td>few 1000</td>
</tr>
<tr>
<td>SimGent</td>
<td>math/d.e.</td>
<td>(some day)</td>
<td>d.e./emul</td>
<td>C or Java</td>
<td>controlled</td>
<td>few 10000</td>
<td></td>
</tr>
</tbody>
</table>

**Recap: Studying Large Distributed HPC Systems**

**Why?** Compare aspects of the possible designs/algorithms/applications
- Response time
- Scalability
- Fault-tolerance
- Throughput
- Robustness
- Fairness

**How?** Several methodological approaches
- Theoretical approach: mathematical study of algorithms
  - Better understanding, impossibility theorems; Everything NP-hard
- Experiments: In vivo: Real applications on Real platforms
  - Believable; Hard and long. Experimental control? Reproducibility?
- Emulation: In vitro: Real applications on Synthetic platforms
  - Better experimental control; Even more difficult
- Simulation: In silico: Prototype of applications on model of systems
  - Simple; Experimental bias

**In Practice?** A lot of tools exist; Some are even usable
- Key trade-off seem to be accuracy vs speed:
  - The more abstract the fastest; The less abstract the most accurate. Really?

---

**Other existing tools**
- Network emulation: ModelNet, DummyNet, ...
  - Tools rather mature, but limited to network
- Applicative emulation: MicroGrid, eWan, Emulab
  - Rarely (never?) used outside the lab where they were created

**Existing evaluation tools for HPC ideas/applications**

<table>
<thead>
<tr>
<th>CPU</th>
<th>Use</th>
<th>Network</th>
<th>Application</th>
<th>Requirement</th>
<th>Settings</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid'5000</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>direct</td>
<td>access</td>
<td>fixed</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>virtualize</td>
<td>access</td>
<td>uncontrolled</td>
</tr>
<tr>
<td>ModelNet</td>
<td>-</td>
<td>-</td>
<td>emulation</td>
<td>emulation</td>
<td>lot material</td>
<td>controlled</td>
</tr>
<tr>
<td>MicroGrid</td>
<td>emulation</td>
<td>-</td>
<td>fine d.e.</td>
<td>emulation</td>
<td>none</td>
<td>controlled</td>
</tr>
<tr>
<td>ns-2</td>
<td>-</td>
<td>-</td>
<td>line d.e.</td>
<td>coarse d.e.</td>
<td>C++/tcl</td>
<td>controlled</td>
</tr>
<tr>
<td>GTENetS</td>
<td>-</td>
<td>-</td>
<td>line d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
</tr>
<tr>
<td>P2PSim</td>
<td>-</td>
<td>-</td>
<td>cste time</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
</tr>
<tr>
<td>PlanetSim</td>
<td>-</td>
<td>-</td>
<td>state machine</td>
<td>Java</td>
<td>controlled</td>
<td>1000000</td>
</tr>
<tr>
<td>ChicSim</td>
<td>coarse d.e.</td>
<td>-</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>C</td>
<td>controlled</td>
</tr>
<tr>
<td>OptSim</td>
<td>coarse d.e.</td>
<td>amount</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
</tr>
<tr>
<td>GridSim</td>
<td>coarse d.e.</td>
<td>math</td>
<td>coarse d.e.</td>
<td>coarse d.e.</td>
<td>Java</td>
<td>controlled</td>
</tr>
<tr>
<td>SimGent</td>
<td>math/d.e.</td>
<td>(some day)</td>
<td>d.e./emul</td>
<td>C or Java</td>
<td>controlled</td>
<td>few 10000</td>
</tr>
</tbody>
</table>

**Recap: Studying Large Distributed HPC Systems**

**Why?** Compare aspects of the possible designs/algorithms/applications
- Response time
- Scalability
- Fault-tolerance
- Throughput
- Robustness
- Fairness

**How?** Several methodological approaches
- Theoretical approach: mathematical study of algorithms
  - Better understanding, impossibility theorems; Everything NP-hard
- Experiments: In vivo: Real applications on Real platforms
  - Believable; Hard and long. Experimental control? Reproducibility?
- Emulation: In vitro: Real applications on Synthetic platforms
  - Better experimental control; Even more difficult
- Simulation: In silico: Prototype of applications on model of systems
  - Simple; Experimental bias

**In Practice?** A lot of tools exist; Some are even usable
- Key trade-off seem to be accuracy vs speed:
  - The more abstract the fastest; The less abstract the most accurate. Really?
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
    ⇒ 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: Realistic ≈ Credible

Accurate
≈ Credible

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

Outline
Experiments for Large-Scale Distributed Systems Research
Methodological Issues
Main Methodological Approaches: In Vivo, In Silico, In Vitro
Existing evaluation tools for HPC ideas / applications
The SimGrid Project
User Interface(s)
MSG: Comparing Heuristics for Concurrent Sequential Processes
GRAS: Developing and Debugging Real Applications
SimDag: Comparing Scheduling Heuristics for DAGs
SMPI: Running MPI applications on top of SimGrid
Models underlying the SimGrid Framework
SimGrid Evaluation
  How accurate?
  How big?
  How fast?
Associated Tools
  Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
  Visualization
Conclusions
Tutorial Recap
Going Further: Experiment planning and Open Science
Take-home Messages

User-visible SimGrid Components
SimGrid user APIs
▶ SimDag: specify heuristics as DAG of (parallel) tasks
▶ MSG: specify heuristics as Concurrent Sequential Processes
  (Java/Ruby/Lua bindings available)
▶ GRAS: develop real applications, studied and debugged in simulator
▶ SMPI: simulate MPI codes
Which API should I choose?
▶ Your application is a DAG \(\sim\) SimDag
▶ You have a MPI code \(\sim\) SMPI
▶ You study concurrent processes, or distributed applications
  ▶ You need graphs about several heuristics for a paper \(\sim\) MSG
  ▶ You develop a real application (or want experiments on real platform) \(\sim\) GRAS
▶ Most popular API (for now): MSG

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
**1. Write the Code of your Agents**

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

**2. Describe your Experiment**

**XML Platform File**
```xml
<?xml version='1.0'?><platform version="2"/>
```

**XML Deployment File**
```xml
<platform version="2">
  <process host="host1" function="master">
    <!-- The master process -->
  </process>
  <process host="host2" function="worker">
    <!-- The workers -->
  </process>
  <process host="host2" function="worker">
    <!-- The workers -->
  </process>
</platform>
```

**3. Glue things together**

```c
int main(int argc, char *argv[]) {
    /* Bind agents' name to their function */
    MSG_function_register("master", &master);
    MSG_function_register("worker", &worker);
    MSG_create_environment("my_platform.xml"); /* Load a platform instance */
    MSG_launch_application("my_deployment.xml"); /* Load a deployment file */
    MSG_main(); /* Launch the simulation */
    INFO1("Simulation took %g seconds", MSG_get_clock());
}
```

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

**Executive summary, but representative**

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

---

**SimGrid in a Nutshell**

**SimGrid Usage Workflow: the MSG example (1/2)**

1. Write the Code of your Agents

   ```c
   int master(int argc, char **argv) {
       for (i = 0; i < number_of_tasks; i++) {
           MSG_task_create(name, comp_size, comm_size, data);
           sprintf(mailbox, "worker-%d", my_id);
           MSG_task_send(t, mailbox);
           MSG_task_receive(atask, my_mailbox);
           MSG_task_destroy(task);    
       }
   }
   ```

2. Describe your Experiment

   **XML Platform File**
   ```xml
   <?xml version='1.0'?><platform version="2"/>
   ```

   **XML Deployment File**
   ```xml
   <platform version="2">
     <process host="host1" function="master">
       <!-- The master process -->
     </process>
     <process host="host2" function="worker">
       <!-- The workers -->
     </process>
     <process host="host2" function="worker">
       <!-- The workers -->
     </process>
   </platform>
   ```

3. Glue things together

   ```c
   int main(int argc, char *argv[]) {
       /* Bind agents' name to their function */
       MSG_function_register("master", &master);
       MSG_function_register("worker", &worker);
       MSG_create_environment("my_platform.xml"); /* Load a platform instance */
       MSG_launch_application("my_deployment.xml"); /* Load a deployment file */
       MSG_main(); /* Launch the simulation */
       INFO1("Simulation took %g seconds", MSG_get_clock());
   }
   ```

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

---

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

**SimGrid in a Nutshell**

**SimGrid is no simulator, but a simulation framework**

---

**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'
[ 0.000] [ Tremblay:master ] Sending 'Task_0' to 'worker-0'
[ 0.148] [ Tremblay:master ] Sending 'Task_1' to 'worker-1'
[ 0.347] [ Tremblay:master ] Sending 'Task_2' to 'worker-2'
[ 0.347] [ Fafard:worker ] Processing 'Task_2'
[ 0.476] [ Tremblay:master ] Sending 'Task_3' to 'worker-0'
[ 0.476] [ Fafard:worker ] Processing 'Task_3'
[ 0.803] [ Fafard:worker ] 'Task_0' done
[ 0.961] [ Fafard:worker ] Sending 'Task_4' to 'worker-1'
[ 0.961] [ Fafard:worker ] 'Task_0' done
[ 1.003] [ Fafard:worker ] 'Task_1' done
[ 1.202] [ Tremblay:master ] Sending 'Task_5' to 'worker-2'
[ 1.202] [ Fafard:worker ] Processing 'Task_4'
[ 1.606] [ Jupiter:worker ] 'Task_2' done
[ 1.606] [ Jupiter:worker ] 'Task_2' done
[ 1.635] [ Fafard:worker ] Processing 'Task_5'
[ 1.637] [ Jupiter:worker ] 'Task_5' done
[ 1.859] [ Fafard:worker ] 'Task_5' done
[ 1.859] [ Fafard:worker ] 'Task_5' done
[ 2.668] [ Tremblay:master ] 'Task_6' done
[ 2.668] [ Fafard:worker ] I'm done. See you!
[ 2.668] [ Fafard:worker ] I'm done. See you!
[ 2.668] [ Fafard:worker ] I'm done. See you!
[ 2.668] [ Fafard:worker ] I'm done. See you!
Simulation time 2.66766
```
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)
- Possible to attach automatic callbacks, or explicitly wait for messages

Differences with MSG
- Messages are typed (+callbacks), where MSG sends raw data chunks
- Socket oriented, where MSG uses mailboxes for rendez-vous
- Code can run in real settings too (so no over-simplification)

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
Communication Performance on a LAN

**Sender:** PPC
**Receiver:** SPARC
**x86**

- 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

---

**Outline**
- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
    - Main Methodological Approaches: In Vivo, In Silico, In Vitro
    - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
- Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
- Associated Tools
  - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping Visualization
- Conclusions
- Tutorial Recap
- Going Further: Experiment planning and Open Science
- Take-home Messages

---

**GRAS eases infrastructure development**

**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 (automatic IDL)
  - Easy to deploy: C ANSI; no dependency; <400kb

---

**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

---

**Research & Development**

**Code**
- API
- GRDK
- GRE
- SimGrid

**With GRAS**
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 ~ until the end)
▶ SD_task.{watch/unwatch}: simulation stops as soon as task's state changes

Full API in the doxygen-generated documentation

SMPI: Running MPI applications on top of SimGrid

Motivations
▶ Reproducible experimentation of MPI code (debugging)
▶ Test MPI code on still-to-build platform (dimensioning)

How it works
▶ smpicc changes MPI calls into SMPI ones (gettimeofday also intercepted)
▶ smpirun starts a classical simulation obeying -hostfile and -np
⇒ Runs unmodified MPI code after recompilation

Implemented calls
▶ Isend; Irecv. Recv; Send; Sendrecv. Wait; Waitall; Waitany. Reduce; Allreduce.
▶ Barrier; Bcast; Reduce; Allreduce (cmd line switch between binary or flat tree)
▶ Comm_size; Comm_rank; Comm_split. Wtime. Init; Finalize; Abort.

Current Work
▶ Implement the rest of the API; Test it more thoroughly
▶ Use it to validate SimGrid at application level (with NAS et Al.)

Outline
● Experiments for Large-Scale Distributed Systems Research
  Methodological Issues
    Main Methodological Approaches: In Vivo, In Silico, In Vitro
    Existing evaluation tools for HPC ideas / applications
● The SimGrid Project
  User Interface(s)
    MSG: Comparing Heuristics for Concurrent Sequential Processes
    GRAS: Developing and Debugging Real Applications
    SimDag: Comparing Scheduling Heuristics for DAGs
    SMPI: Running MPI applications on top of SimGrid
  Models underlying the SimGrid Framework
  SimGrid Evaluation
    How accurate?
    How big?
    How fast?
  Associated Tools
    Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
    Visualization
● Conclusions
  Tutorial Recap
  Going Further: Experiment planning and Open Science
  Take-home Messages

Analytic Models underlying the SimGrid Framework

Main challenges for SimGrid design
▶ Simulation accuracy:
  ▶ Designed for HPC scheduling community ~ 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: \( \text{Time} = L + \frac{\text{size}}{B} \)
- Resource work at given rate (\( B \), in MFlop/s or Mb/s)
- Each use have a given latency (\( L \), in s)

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

#### Modeling Single-Hop Networks
- **Simplistic:** \( T = \lambda + \frac{\text{size}}{\beta} \);
- **More accurate:** [Padhye, Firoiu, Towsley, Krusoe 2000]

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

- \( \rho \): loss indication rate
- \( b \): #packages acknowledged per ACK
- \( T_0 \): TCP average retransmission timeout value
- \( \rho \) and \( b \) not known in general (model hard to instanciable)
- Let’s keep instanciable: use \( \beta' = \min(\beta, \frac{W_{\text{max}}}{\text{RTT}}) \) (TCP windowing)

### Analytical Network Models

TCP bandwidth sharing studied by several authors
- Study the path of each and every network packet
- Remember your networking class?

#### Notations
- \( \mathcal{L} \): set of links
- \( C_i \): capacity of link \( i \) (\( C_i > 0 \))
- \( n_i \): amount of flows using link \( i \)

#### Feasibility constraint
- Links deliver their capacity at most:

\[
\forall l \in \mathcal{L}, \sum_{f \ni l} \lambda_f \leq C_i
\]

### Max-Min Fairness

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

#### Bottleneck links
- For each flow \( f \), one of the links is the limiting one \( i \)
- The objective function gives that \( i \) is saturated, and \( f \) gets the biggest share

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

---

Martin Quinson  Experimenting HPC Systems with Simulation  The SimGrid Project  (37/72)

---

Max-Min Fairness on Homogeneous Linear Network

$C_l$: capacity of link $l$; $n_i$: amount of flows using $l$; $\lambda_f$: transfer rate of $f$.

**Algorithm:** loop on these steps

- search for the bottleneck link: share of its flows (ie, $\frac{C_l}{n_i}$) is minimal
- set all flows using it
- remove the link

Homogeneous Linear Network

**Link 0:**

$C_0 = C, n_1 = 2$

$\lambda_0 \leftarrow C/2$

**Link 1:**

$C_1 = C, n_2 = 2$

$\lambda_1 \leftarrow C/2$

**Link 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

Side note: OptorSim 2.1 on Backbone

OptorSim (developed @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: $\frac{C_l}{n_i}$

2. For each flow, compute what it gets: $\lambda_f = \min_{l \in f} \left( \frac{C_l}{n_i} \right)$

<table>
<thead>
<tr>
<th>Link</th>
<th>Flow 0</th>
<th>Flow 1</th>
<th>Flow 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$C_1$</td>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$C_2$</td>
<td>1000</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$C_3$</td>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$C_4$</td>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

$\lambda_1$ limited by link 2, but 499 still unused on link 2

This “unwanted feature” is even listed in the README file...

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

Availability traces are just events
t_0 \rightarrow 100%, t_1 \rightarrow 50%, t_2 \rightarrow 80%, etc.

Also qualitative state changes (on/off)
**Validation experiments on a single link**

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

**Evaluation Results**
- Packet-level tools don’t completely agree
- SSFNet TCP_FAST_INTERVAL bad default
- GTNetS is equally distant from others
- Old SimGrid model omitted slow start effects
  - Statistical analysis of GTNetS slow-start
  - Better instantiation of MaxMin model
  - $\beta'' \sim 0.92 \times \beta'$; $\lambda \sim 10.4 \times \lambda$
- Resulting validity range quite acceptable
  - $|\epsilon| \leq 12\% \approx 162\%$
  - $S < 100KB$
  - $S > 100KB$
- $|\epsilon_{\text{max}}| \leq 6\%$

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

Maybe the error is not SimGrid's:
- Big error because GTNetS multi-phased
- Seen the same in NS3, emulation, ...
- Phase Effect: Periodic and deterministic traffic may resonate [Floyd & Jacobson 91]
- Impossible in Internet (thx random noise)

We’re adding random jitter to continue SimGrid validation
Simulation scalability assessment (how big?)

Master/Workers on amd64 with 4Gb

<table>
<thead>
<tr>
<th>#tasks</th>
<th>Context mechanism</th>
<th>100</th>
<th>500</th>
<th>1,000</th>
<th>5,000</th>
<th>10,000</th>
<th>25,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ucontext</td>
<td>0.16</td>
<td>0.19</td>
<td>0.23</td>
<td>0.42</td>
<td>0.74</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>0.41</td>
<td>0.59</td>
<td>0.94</td>
<td>7.6</td>
<td>27.3</td>
<td>58.3</td>
</tr>
<tr>
<td>1,000</td>
<td>ucontext</td>
<td>0.76</td>
<td>0.53</td>
<td>0.83</td>
<td>1.3</td>
<td>1.77</td>
<td>4.54</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>1.6</td>
<td>1.9</td>
<td>2.3</td>
<td>13</td>
<td>40.4</td>
<td>1.66</td>
</tr>
<tr>
<td>10,000</td>
<td>ucontext</td>
<td>3.7</td>
<td>3.8</td>
<td>4.5</td>
<td>9.5</td>
<td>34.5</td>
<td>90.7</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>14</td>
<td>13</td>
<td>15</td>
<td>26</td>
<td>77</td>
<td>220</td>
</tr>
<tr>
<td>100000</td>
<td>ucontext</td>
<td>42</td>
<td>44</td>
<td>46</td>
<td>50</td>
<td>52.3</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>java</td>
<td>121</td>
<td>130</td>
<td>134</td>
<td>163</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

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

These results are old already

v3.3.3 is 30% faster

v3.3.4 leads to lazy evaluation

Extensibility with UNIX contextes

<table>
<thead>
<tr>
<th>#tasks</th>
<th>Size</th>
<th>25,000</th>
<th>50,000</th>
<th>100,000</th>
<th>200,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>128KB</td>
<td>1.6</td>
<td>0.5</td>
<td>0.5</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>12KB</td>
<td>2</td>
<td>0.8</td>
<td>1.2</td>
<td>3.5</td>
</tr>
<tr>
<td>10,000</td>
<td>128KB</td>
<td>5.5</td>
<td>5.7</td>
<td>4.1</td>
<td>6.7</td>
</tr>
<tr>
<td></td>
<td>12KB</td>
<td>33</td>
<td>161</td>
<td>167</td>
<td>161</td>
</tr>
<tr>
<td>1,00000</td>
<td>128KB</td>
<td>208</td>
<td>161</td>
<td>167</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>12KB</td>
<td>33</td>
<td>161</td>
<td>167</td>
<td>161</td>
</tr>
</tbody>
</table>

*: out of memory

Simulation scalability assessment (how fast?)

During Summer 2009, 2 interns @CERN evaluated grid simulators

- Attempted to simulate one day on their data grid (1.5 million file transfers)
- Their final requirements:
  - Basic processing induce 30M operations daily
  - User requests induce \( \approx 2M \) operations daily
  - Evaluations should consider one month of operation

Findings

- Experiment to be redone?
- Controlled experimental settings
- With recent versions of the tools
- More metrics
- Better if not done by us (you?)

Simulation is only one piece of the workflow

- Needed Input:
  - Platform (quantitative and qualitative)
  - Application (code and deployment; workload)
- Provided Output: Text logs, Graphical Visualization

Platform Instantiation

To use a simulator, one must instantiate the models

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
- Synthetic platform generator: use random generators
- Automatic mapping: automated tomography tool
**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

---

**Example of XML file**

```xml
<?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"/>
    </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 ∈ \{shared, fatpipe\} (fatpipe → no sharing)
- Declare routes from each host to each host (list of links)
- Arbitrary data can be attached to components using the <prop> tag

---

**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](http://pda.gforge.inria.fr)

(+ tool to extract platform subsets)

---

**Synthetic Topology Generation**

**Designing a Realistic Platform Generator**

- Examine real platforms; Discover principles; Implement a generator
- Subject of studies in Networking for years ⇒ Loads of generation methods

**Simulacrum**: Generic GUI to generate SimGrid platform files

**Selecting from existing**

- Creating + Filtering

**Other tools**

- Several well known generators for networking community, eg Brite
- Grid-G: All in one framework, Grid specific
The SimGrid Project (59/72)

Automatic Network Mapping

Main Issue of synthetic generators: **Realism!**
- **Solution:** Actually map a real platform
- **Tomography:** 2-steps process (end-to-end Measurements; Reconstruct a graph)

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

The ALNeM project (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
- **Project started in 2002, still underway 😊**

Measurement step
- Network level tools (BGP, SNMP, ICMP)
  - use restricted for security reason
  - hard to get a App-Level view from them
- We rely on simple E2E measurements (latency/bandwidth)

Evaluation methodology

How to evaluate the reconstruction algorithms’ quality?

Several evaluation metrics
1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:
   \[ \text{Interf} \left( (a, b), (c, d) \right) = 1 \iff \frac{BW(a \rightarrow b)}{BW(a \rightarrow b \parallel c \rightarrow d)} \approx 2 \]
3. Compare application running times (application-level)

<table>
<thead>
<tr>
<th>Comm. schema</th>
<th>// comm</th>
<th># steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token-ring</td>
<td>Ring</td>
<td>No</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Tree</td>
<td>No</td>
</tr>
<tr>
<td>All2All</td>
<td>Clique</td>
<td>Yes</td>
</tr>
<tr>
<td>2D</td>
<td>Yes</td>
<td>√/procs</td>
</tr>
</tbody>
</table>

(Other) Methodological Challenge
- **Goal:** Quantify similarity between initial and reconstructed platforms
- **Impossible to test against real platform:** Reconstructed platform doesn’t exist
- **Testing on simulator:** both initial and reconstructed platforms are simulated

Leveraging GRAS framework (of course)

Experiments on simulator: Renater platform
- **Real platform built manually (real measurements + admin feedback)**

End to end

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

Measurement step
- **Network level tools (BGP, SNMP, ICMP)**
  - use restricted for security reason
  - hard to get a App-Level view from them
- We rely on simple E2E measurements (latency/bandwidth)

Evaluation criteria
- **Clique:**
  - End-to-end meas.
  - Interference count
  - Application-level

Accuracy
- **Clique:**
  - Correct pred.
  - False pos.
  - False neg.

Application-level
- **Clique:**
  - Aggregate accuracy discutable

Evaluation methodology

How to evaluate the reconstruction algorithms’ quality?

Several evaluation metrics
1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:
   \[ \text{Interf} \left( (a, b), (c, d) \right) = 1 \iff \frac{BW(a \rightarrow b)}{BW(a \rightarrow b \parallel c \rightarrow d)} \approx 2 \]
3. Compare application running times (application-level)

<table>
<thead>
<tr>
<th>Comm. schema</th>
<th>// comm</th>
<th># steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token-ring</td>
<td>Ring</td>
<td>No</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Tree</td>
<td>No</td>
</tr>
<tr>
<td>All2All</td>
<td>Clique</td>
<td>Yes</td>
</tr>
<tr>
<td>2D</td>
<td>Yes</td>
<td>√/procs</td>
</tr>
</tbody>
</table>

(Other) Methodological Challenge
- **Goal:** Quantify similarity between initial and reconstructed platforms
- **Impossible to test against real platform:** Reconstructed platform doesn’t exist
- **Testing on simulator:** both initial and reconstructed platforms are simulated

Leveraging GRAS framework (of course)

Experiments on simulator: Renater platform
- **Real** platform built manually (real measurements + admin feedback)

End to end

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

Measurement step
- **Network level tools (BGP, SNMP, ICMP)**
  - use restricted for security reason
  - hard to get a App-Level view from them
- We rely on simple E2E measurements (latency/bandwidth)

Evaluation criteria
- **Clique:**
  - End-to-end meas.
  - Interference count
  - Application-level

Accuracy
- **Clique:**
  - Correct pred.
  - False pos.
  - False neg.

Application-level
- **Clique:**
  - Aggregate accuracy discutable

Evaluation methodology

How to evaluate the reconstruction algorithms’ quality?

Several evaluation metrics
1. Compare end-to-end measurements (communication-level)
2. Compare interference amount:
   \[ \text{Interf} \left( (a, b), (c, d) \right) = 1 \iff \frac{BW(a \rightarrow b)}{BW(a \rightarrow b \parallel c \rightarrow d)} \approx 2 \]
3. Compare application running times (application-level)

<table>
<thead>
<tr>
<th>Comm. schema</th>
<th>// comm</th>
<th># steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token-ring</td>
<td>Ring</td>
<td>No</td>
</tr>
<tr>
<td>Broadcast</td>
<td>Tree</td>
<td>No</td>
</tr>
<tr>
<td>All2All</td>
<td>Clique</td>
<td>Yes</td>
</tr>
<tr>
<td>2D</td>
<td>Yes</td>
<td>√/procs</td>
</tr>
</tbody>
</table>

(Other) Methodological Challenge
- **Goal:** Quantify similarity between initial and reconstructed platforms
- **Impossible to test against real platform:** Reconstructed platform doesn’t exist
- **Testing on simulator:** both initial and reconstructed platforms are simulated

Leveraging GRAS framework (of course)

Experiments on simulator: Renater platform
- **Real** platform built manually (real measurements + admin feedback)

End to end

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

Measurement step
- **Network level tools (BGP, SNMP, ICMP)**
  - use restricted for security reason
  - hard to get a App-Level view from them
- We rely on simple E2E measurements (latency/bandwidth)

Evaluation criteria
- **Clique:**
  - End-to-end meas.
  - Interference count
  - Application-level

Accuracy
- **Clique:**
  - Correct pred.
  - False pos.
  - False neg.

Application-level
- **Clique:**
  - 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**

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Clique</th>
<th>TestBW</th>
<th>TestLat</th>
<th>ImpTreeBW</th>
<th>ImpTreeLat</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bandwidth</td>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Application-level measurements**

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Clique</th>
<th>TestBW</th>
<th>TestLat</th>
<th>ImpTreeBW</th>
<th>ImpTreeLat</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bandwidth</td>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Clique</th>
<th>TestBW</th>
<th>TestLat</th>
<th>ImpTreeBW</th>
<th>ImpTreeLat</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bandwidth</td>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Application-level measurements**

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Clique</th>
<th>TestBW</th>
<th>TestLat</th>
<th>ImpTreeBW</th>
<th>ImpTreeLat</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bandwidth</td>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Interpretation**

- None of the proposed heuristic is satisfactory
- **Future work:** improve this! Becomes really tricky. Maybe data-mining issue?

**Outline**

- **Experiments for Large-Scale Distributed Systems Research**
  - Methodological Issues
  - Main Methodological Approaches: In Vivo, In Silico, In Vitro
  - Existing evaluation tools for HPC ideas / applications
- **The SimGrid Project**
  - User Interface(s)
    - MSG: Comparing Heuristics for Concurrent Sequential Processes
    - GRAS: Developing and Debugging Real Applications
    - SimDag: Comparing Scheduling Heuristics for DAGs
    - SMPI: Running MPI applications on top of SimGrid
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
    - How accurate?
    - How big?
    - How fast?
  - Associated Tools
    - Platform Instantiation: Catalog, Synthetic Generation, Network Mapping
  - Visualization
- **Conclusions**
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

**Visualizing SimGrid Simulations with Trivia**

Simulations can produce a *lot* of logs

- Everyone produces ad-hoc parsing scripts
- Not always easy, graphically visualizing more appealing

**Building the right visualization tool**

- Easy to build a *demoware*: fancy but not really useful
- Trivia: separate (established) project; SimGrid only produces adapted traces
- Events, Tasks can be given a application-level semantic category
- Still ongoing effort (integrated in stable releases since spring only)
Outline

- Experiments for Large-Scale Distributed Systems Research
  - Methodological Issues
    - Main Methodological Approaches: In Vivo, In Silico, In Vitro
    - Existing evaluation tools for HPC ideas / applications
- The SimGrid Project
  - User Interface(s)
  - Models underlying the SimGrid Framework
  - SimGrid Evaluation
  - Associated Tools
- Conclusions
  - Tutorial Recap
  - Going Further: Experiment planning and Open Science
  - Take-home Messages

Conclusions on Distributed Systems Research

Research on Large-Scale Distributed Systems

- Reflexion about common methodologies needed (reproducible results needed)
- Purely theoretical works limited (simplistic settings ~ 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 available, 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; Java, Lua, Ruby bindings
  - Also trace-driven simulations (user-defined events and callbacks)

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

SMPI: Running MPI applications on top of SimGrid (beta quality)
  - Runs unmodified MPI code after recompilation (still partial implementation)

Other interfaces possible: OpenMP, BSP-like (any volunteer?)

SimGrid is an active and exciting project

Future Plans

- Better usability: build around simulator
  (statistics tools, campaign management)
- Extreme Scalability for P2P
- Model-checking and semantic debugging
- Emulation solution à la MicroGrid

Large community

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

- 100 subscribers to the user mailing list (40 to -devel)
- 70 scientific publications using the tool for their experiments
- 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!
Grid Simulation and Open Science

Requirement on Experimental Methodology (what do we want)
▶ Standard methodologies and tools: Grad students learn them to be operational
▶ Incremental knowledge: Read a paper, Reproduce its results, Improve.
▶ Reproducible results: Compare easily experimental scenarios
  Reviewers can reproduce result, Peers can work incrementally (even after long time)

Current practices in the field (what do we have)
▶ Very little common methodologies and tools; many home-brewed tools
▶ Experimental settings rarely detailed enough in literature

These issues are tackled by the SimGrid community
▶ Released, open-source, stable simulation framework
▶ Extensive optimization and validation work
▶ Separation of simulated application and experimental conditions
▶ Are we there yet? Not quite

SimGrid and Open Science

Simulations are reproducible ... provided that authors ensure that
▶ Need to publish source code, platform file, statistic extraction scripts . . .
▶ Almost no one does it. I don’t (shame, shame). Why?

Technical issues to tackle
▶ Archiving facilities, Versioning, Branch support, Dependencies management
▶ Workflows automating execution of test campaigns (myexperiment.org)
▶ We already have most of them (Makefiles, Maven, debs, forges, repositories, . . .)
▶ But still, we don’t use it. Is the issue really technical?

Sociological issues to tackle
▶ A while ago, simulators were simple, only filling gant charts automatically
▶ We don’t have the culture of reproducibility:
  ▶ “My scientific contribution is the algorithm, not the crappy demo code”
  ▶ But your contribution cannot be assessed if it cannot be reproduced!
▶ I don’t have any definitive answer about how to solve it

Building Open Science Around the Simulator

Going further toward Open Science
▶ Issues we face in simulation are common to every experimental methodologies
  Test planning, Test Campaign Management, Statistic Extraction
▶ Tool we need to help Open Science arise in simulation would help others
▶ Why not step back and try to unit efforts?

What would a perfect world look like?
A single simulation using SimGrid

Take-home Messages

HPC and Grid applications tuning and assessment
▶ Challenging to do; Methodological issues often neglected
▶ Several methodological ways: in vivo, in vitro, in silico; none perfect

The SimGrid Simulation Framework
▶ Mature Framework: validated models, software quality assurance
▶ You should use it!

We only scratched the corner of the problem
▶ A single simulation is just a brick of the scientific workflow
  ▶ We need more associated tools for campaign management, etc.
▶ Open Science is a must! (please don’t say the truth to physicians or biologists)
  ▶ Technical issues faced, but even more sociological ones
  ▶ Solve it not only for simulation, but for all methodologies at the same time

We still have a large amount in front of us ☺