Grid’5000

a testbed for reproducible research on HPC, Clouds, Big Data and Networking

Lucas Nussbaum

With the Grid’5000 architects committee and the Grid’5000 technical team
Distributed computing: a peculiar field in CS

- Most contributions are validated using experiments
  - Very little formal validation
  - Even for theoretical work ~ simulation (SimGrid)

- Performance and scalability are central to results
  - But depend greatly on the environment (hardware, network, software stack, etc.)
  - Many contributions are about *fighting* the environment (load balancing, fault tolerance, middlewares, etc.)

- Experimenting is difficult and time-consuming

- Shifts the scope for reproducible research:
  - How can one perform *reproducible* experiments?
  - Very similar to (not computational) biology or physics
Research pipeline in experimental DC research

the easy part

the HARD part

Measured Data

Analytic Data

Computational Results

Presentation Code

Figures

Tables

Numerical Summaries

Published Article

Text

Experiment Code (workload injector, VM recipes, ...)

Protocol (Design of Experiments)

Scientific Question

Nature/System/...

Analytic Code

Processing Code

Author

Analysis/experiment feedback loop

Reader

Lucas Nussbaum  Grid'5000
The Grid’5000 testbed

- World-leading testbed for distributed computing
  - 10 sites, 25 clusters, 1000 nodes, 8000 cores
  - Dedicated 10-Gbps backbone network
  - 550 users and 100 publications per year
The Grid’5000 testbed

- World-leading testbed for distributed computing
  - 10 sites, 25 clusters, 1000 nodes, 8000 cores
  - Dedicated 10-Gbps backbone network
  - 550 users and 100 publications per year

- Not a typical grid / cluster / Cloud, more a meta-grid, meta-cloud:
  - Used by CS researchers in HPC / Clouds / Big Data / Networking to perform experiments

- Design goals:
  - Large-scale, shared infrastructure
  - Support high-quality, reproducible research
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
Description and verification of the environment

Typical needs:

- How can I find suitable resources for my experiment?
- How sure can I be that the actual resources will match their description?
- What was the hard drive on the nodes I used six months ago?
Description and verification of the environment

Typical needs:
► How can I find suitable resources for my experiment?
► How sure can I be that the actual resources will match their description?
► What was the hard drive on the nodes I used six months ago?

Selection and reservation of resources (OAR)
Description of resources (Reference API)
Verification of resources (g5k-checks)

OAR properties
nodes description

API requests

OAR commands and API requests
Users
High-level tools
Description and selection of resources

- Describing resources → understand results
  - Detailed description on the Grid’5000 wiki
  - Machine-parsable format (JSON)
  - Archived (State of testbed 6 months ago?)

```json
"processor": {
  "cache_l2": 9388603,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": ",",
  "other_description": ",",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l2": null,
  "cache_l1": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-1",
"type": "node",
"architecture": {
  "platform_type": "x86_64",
  "smt_size": 4,
  "smp_size": 1
},
"main_memory": {
  "ram_size": 17179869184,
  "virtual_size": null
},
"storage_devices": [
  {
    "model": "Hitachi HDS72103",
    "size": 298023223876.953,
    "driver": "ahci",
    "interface": "SATA II",
    "rev": "JMF0",
    "device": "sda"
  }
],
```
Description and selection of resources

▶ Describing resources ～ understand results

♦ Detailed description on the Grid’5000 wiki
♦ Machine-parsable format (JSON)
♦ Archived (State of testbed 6 months ago?)

▶ Selecting resources

♦ OAR database filled from JSON

```
oarsub -p "wattmeter=’YES’ and gpu=’YES’"
oarsub -l "cluster=’a’/nodes=1+cluster=’b’ and eth10g=’Y’/nodes=2,walltime=2"
```

```json
"processor": {
  "cache_l2": 3388609,
  "cache_l1": null,
  "model": "Intel Xeon",
  "instruction_set": "",
  "other_description": "",
  "version": "X3440",
  "vendor": "Intel",
  "cache_l3": null,
  "clock_speed": 2530000000.0
},
"uid": "graphene-l",
"type": "node",
"architecture": {
  "platform_type": "x86_64",
  "smt_size": 4,
  "smp_size": 1
},
"main_memory": {
  "ram_size": 17179869184,
  "virtual_size": null
},
"storage_devices": [
  {
    "model": "Hitachi HDS72103",
    "size": 2980232232875.953,
    "driver": "ahci",
    "interface": "SATA II",
    "rev": "JPP0",
    "device": "sda"
  }
],
```
Verification of resources

- Inaccuracies in resources descriptions → dramatic consequences:
  ♦ Mislead researchers into making false assumptions
  ♦ Generate wrong results → retracted publications!

- Happen frequently: maintenance, broken hardware (e.g. RAM)
Verification of resources

- Inaccuracies in resources descriptions → dramatic consequences:
  - Mislead researchers into making **false assumptions**
  - Generate **wrong results** → retracted publications!

- Happen frequently: maintenance, broken hardware (e.g. RAM)

- Our solution: **g5k-checks**
  - Runs at node boot (can also be run manually by users)
  - Retrieves current description of node in Reference API
  - Acquire information on node using OHAI, ethtool, etc.
  - Compare with Reference API

Future work
- Verification of performance, not just availability and configuration of hardware (hard drives, network, etc.)
- Provide tools to capture the state of the testbed; archival with the rest of the experiment's data
Verification of resources

- Inaccuracies in resources descriptions → dramatic consequences:
  - Mislead researchers into making false assumptions
  - Generate wrong results → retracted publications!

- Happen frequently: maintenance, broken hardware (e.g. RAM)

- Our solution: g5k-checks
  - Runs at node boot (can also be run manually by users)
  - Retrieves current description of node in Reference API
  - Acquire information on node using OHAI, ethtool, etc.
  - Compare with Reference API

- Future work
  - Verification of performance, not just availability and configuration of hardware (hard drives, network, etc.)
  - Provide tools to capture the state of the testbed → archival with the rest of the experiment’s data
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
Reconfiguring the testbed

- Typical needs:
  - How can I install $SOFTWARE on my nodes?
  - How can I add $PATCH to the kernel running on my nodes?
  - Can I run a custom MPI to test my fault tolerance work?
  - How can I experiment with that Cloud/Grid middleware?
  - Can I get a stable (over time) software environment for my experiment?
Reconfiguring the testbed

- Typical needs:
  - How can I install $SOFTWARE on my nodes?
  - How can I add $PATCH to the kernel running on my nodes?
  - Can I run a custom MPI to test my fault tolerance work?
  - How can I experiment with that Cloud/Grid middleware?
  - Can I get a stable (over time) software environment for my experiment?

- Likely answer on any production facility: you can’t

- Or: use virtual machines ↼ experimental bias (performance), limitations
Reconfiguring the testbed

- Operating System reconfiguration with Kadeploy:
  - Provides a *Hardware-as-a-Service* Cloud infrastructure
  - Enable users to deploy their own software stack & get *root* access
  - **Scalable, efficient, reliable and flexible:** 200 nodes deployed in ~5 minutes (120s with Kexec)

- Customize networking environment with KaVLAN
  - Deploy intrusive middlewares (Grid, Cloud)
  - Protect the testbed from experiments
  - Avoid network pollution
  - By reconfiguring VLANS \(\sim\) almost no overhead
  - Recent work: support several interfaces

---

**Kadeploy**

**default VLAN**
- routing between Grid'5000 sites

**global VLANS**
- all nodes connected at level 2, no routing

**local, isolated VLAN**
- only accessible through a SSH gateway connected to both networks

**routed VLAN**
- separate level 2 network, reachable through routing

Lucas Nussbaum  Grid'5000
Creating and sharing Kadeploy images

- Avoid manual customization:
  - Easy to forget some changes
  - Difficult to describe
  - The full image must be provided
  - Cannot really reserve as a basis for future experiments (similar to binary vs source code)

- Kameleon: Reproducible generation of software appliances
  - Using recipes (high-level description)
  - Persistent cache to allow re-generation without external resources (Linux distribution mirror) \(\sim\) self-contained archive
  - Supports Kadeploy images, LXC, Docker, VirtualBox, qemu, etc.

http://kameleon.imag.fr/
Changing experimental conditions

- Reconfigure experimental conditions with Distem
  - Introduce heterogeneity in an homogeneous cluster
  - Emulate complex network topologies

CPU cores

<table>
<thead>
<tr>
<th>CPU performance</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VN 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VN 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual node 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://distem.gforge.inria.fr/
Testing Charm++ load balancing with Distem

No load balancing

- total run time: 473s
- Average CPU usage: 51%

RefineLB

- total run time: 443s
- Average CPU usage: 59%

- Every 2 minutes, 1/8 of the nodes are downclocked for 2 minutes
- On the figure, node 0 has been downclocked
- Visible improvement thanks to load balancing
What else can we enable users to change?

- BIOS settings
  - Power management settings
  - CPU features (Hyperthreading, Turbo mode, etc.)

- Cooling system \(\sim\) temperature in the machine room?
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
Monitoring experiments

Goal: enable users to understand what happens during their experiment

- System-level probes (usage of CPU, memory, disk, with Ganglia)
- Infrastructure-level probes
  - Network, power consumption
  - Captured at high frequency (1 Hz)
  - Live visualization
  - REST API
  - Long-term storage

[Graphs showing network and power consumption data]
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
Improving control and description of experiments

- Legacy way of performing experiments: shell commands
  - 😞 time-consuming
  - 😞 error-prone
  - 😞 details tend to be forgotten over time

- Promising solution: automation of experiments
  - ℹ️ Executable description of experiments

- Support from the testbed: Grid’5000 RESTful API
  (Resource selection, reservation, deployment, monitoring)
Tools for automation of experiments

Several projects around Grid’5000 (but not specific to Grid’5000):

- **g5k-campaign** (Grid’5000 tech team)
- **Expo** (Cristian Ruiz)
- **Execo** (Mathieu Imbert)
- **XPFlow** (Tomasz Buchert)

Features:

- Facilitate scripting of experiments in high-level languages (Ruby, Python)
- Provide useful and efficient abstractions:
  - Testbed management
  - Local & remote execution of commands
  - Data management
- *Engines* for more complex processes

---

**XPFlow**

**Experiment description and execution as a Business Process Workflow**

Supports parallel execution of activities, error handling, snapshotting, built-in logging and provenance collection, etc.

```python
engine.process :exp do |site, switch|
  s = run g5k.switch, site, switch
  ns = run g5k.nodes, s
  r = run g5k.reserve_nodes,
      :nodes => ns, :time => '2h',
      :site => site, :type => :deploy
  master = (first_of ns)
  rest = (tail_of ns)
  run g5k.deploy,
      r, :env => 'squeeze-x64-nfs'
  checkpoint :deployed
  parallel :retry => true do
    forall rest do |slave|
      run :install_pkgs, slave
    end
  end
  sequence do
    run :install_pkgs, master
    run :build_netgauge, master
    run :dist_netgauge,
         master, rest
    end
  end
  checkpoint :prepared
  output = run :netgauge, master, ns
  checkpoint :finished
  run :analysis, output, switch
end
```
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
Realis: evaluating current experimental practices

- COMPAS: Conférence en Parallélisme, Architecture et Système
  - French-speaking, mostly for PhD students

- Realis: test reproducibility of papers submitted to COMPAS
  - Participating authors submit their experimentation description
  - Each author reproduces the experiments from another article
    - Try to get identical results, without contacting the authors
    - Evaluate the quality (flexibility, robustness) of the approach

- Most results can be reproduced (but none without contacting the authors)
Outline

1. Introduction
2. Description and verification of the environment
3. Reconfiguring the testbed to meet experimental needs
4. Monitoring experiments, extracting and analyzing data
5. Improving control and description of experiments
6. Evaluating current experimental practices
7. Conclusions
A multi-tier challenge

Experimental methodology:
experiment design & planning (workflow) ; description of scenarios, of experimental conditions ; definition of metrics ; analysis and visualization of results

Orchestration of experiments:
organize the execution of complex and large-scale experiments (workflow) ; run experiments unattended and efficiently ; handles failures ; compose experiments

Basic services: common tools required by most experiments

Interact w/ testbed
find, reserve and configure resources

Test resources before using them

Manage the environment
Control a large number of nodes

Manage data
Change experimental conditions

Instrument the application & the environment
Monitoring and data collection

Experimental testbed (e.g Grid’5000, FutureGrid):
reconfigurable hardware and network; isolation; some instrumentation and monitoring
Conclusions

- Grid’5000: a testbed for high-quality, reproducible research on HPC, Clouds, Big Data and Networking

- With a unique combination of features
  - Description and verification of testbed
  - Reconfiguration (hardware, network)
  - Monitoring
  - Support for automation of experiments

- Paving the way to Open Science of HPC and Cloud – mid term goals:
  - Fully automated execution of experiments ($\approx$ CI in SE)
  - Automated tracking + archiving of experiments and associated data

- Try it yourself! $\rightsquigarrow$ Open Access program
Bibliography

- **Resources management:** Resources Description, Selection, Reservation and Verification on a Large-scale Testbed. [http://hal.inria.fr/hal-00965708](http://hal.inria.fr/hal-00965708)

- **Kadeploy:** Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters. [http://hal.inria.fr/hal-00909111](http://hal.inria.fr/hal-00909111)

- **KaVLAN, Virtualization, Clouds deployment:**
  - Adding Virtualization Capabilities to the Grid'5000 testbed. [http://hal.inria.fr/hal-00946971](http://hal.inria.fr/hal-00946971)
  - Enabling Large-Scale Testing of IaaS Cloud Platforms on the Grid'5000 Testbed. [http://hal.inria.fr/hal-00907888](http://hal.inria.fr/hal-00907888)

- **Kameleon:** Reproducible Software Appliances for Experimentation. [https://hal.inria.fr/hal-01064825](https://hal.inria.fr/hal-01064825)

- **Distem:** Design and Evaluation of a Virtual Experimental Environment for Distributed Systems. [https://hal.inria.fr/hal-00724308](https://hal.inria.fr/hal-00724308)

- **XP management tools:**
  - A survey of general-purpose experiment management tools for distributed systems. [https://hal.inria.fr/hal-01087519](https://hal.inria.fr/hal-01087519)
  - XPFlow: A workflow-inspired, modular and robust approach to experiments in distributed systems. [https://hal.inria.fr/hal-00909347](https://hal.inria.fr/hal-00909347)
  - Using the EXECO toolbox to perform automatic and reproducible cloud experiments. [https://hal.inria.fr/hal-00861886](https://hal.inria.fr/hal-00861886)
  - Expo: Managing Large Scale Experiments in Distributed Testbeds. [https://hal.inria.fr/hal-00953123](https://hal.inria.fr/hal-00953123)

- **Kwapi:** A Unified Monitoring Framework for Energy Consumption and Network Traffic. [https://hal.inria.fr/hal-01167915](https://hal.inria.fr/hal-01167915)

- **Realis’2014:** Reproductibilité expérimentale pour l’informatique en parallélisme, architecture et système. [https://hal.inria.fr/hal-01011401](https://hal.inria.fr/hal-01011401)